Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

OPTIMIZING DEEP DREAM

Autor: Eamon Joseph Thornton Powell
Tutor: Eduardo Garrido Merchán
Ponente: Daniel Hernández Lobato

Junio 2018

OPTIMIZING DEEP DREAM

Autor: Eamon Joseph Thornton Powell
Tutor: Eduardo Garrido Merchán
Ponente: Daniel Hernández Lobato

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Resumen

Resumen

En este estudio se busca optimizar la herramienta de transferencia de estilo mediante redes neuronales convolucionales denominada **Deep Style**. A través de la red convolucional y dos imágenes de entrada arbitrarias es capaz de sintetizar una tercera imagen que captura atributos de las dos imágenes de entrada en distintos niveles de abstracción, resultado en una fusión entre en contenido de una de las imágenes, denominada imagen de contenido; y el estilo de la otra, denominada imagen de estilo.
Desafortunadamente, la diversidad en el contenido de las imágenes que recibe esta herramienta a menudo produce resultados poco atractivos que, combinados con el tiempo de generación de cada imagen, pueden suponer una experiencia frustrante para el usuario. Por ello, a través de modelos de regresión y optimización por búsqueda aleatoria se busca mejorar la selección de parámetros para así brindar al usuario mejores configuraciones personalizadas específicamente para la imagen que quiera utilizar.
Previamente al desarrollo de este sistema de recomendación se seleccionará una de entre numerosas implementaciones de la herramienta **Deep Style**, se estudiará en profundidad su funcionamiento y tras conocer su estructura, se desarrollará acordemente una metodología que se adecue a la arquitectura de la misma.
A su vez se contará con un conjunto de imágenes, que servirán como grupo de control, a partir de las cuales se realizará la generación de imágenes para pruebas y procesos de evaluación.
Tras realizar la fase de optimización y selección de modelo se completará el estudio realizando unos experimentos para validar el éxito del proceso. Por último se discutirán los resultados obtenidos frente a los objetivos, hipótesis y predicciones formuladas al inicio del desarrollo y se formularán posibles maneras de mejorar o expandir este proyecto.

Palabras Clave

Deep Style, Red neuronal convolucional, aprendizaje automático, optimización, Deep Dream, Torch, transferencia de estilo, modelos de regresión.

Abstract

The purpose of this research is to optimize a style transfer tool that uses convolutional neural networks called **Deep Style**. Through convolutional neural networks and the use of two arbitrary input images this tool is capable of synthesizing a third image that represents the attributes of both input images at different levels of abstraction, resulting in the fusion between the content of one of the images referred to as the content image; and the style of the other, referred to as the style image.
Unfortunately, the variety of content in the input images of this tool often yields poor results which, combined with the amount of time needed to generate each image, can make for a frustrating experience for the user. For this reason, through regression models and random search optimization, we aim to improve parameter selection to offer the user better configurations custom-made for the input image of choice.
Before the development section of this project, one of many available **Deep Style** implementations will be chosen to be thoroughly analyzed. After getting acquainted with its structure, a methodology that properly adapts to its architecture will be implemented.
On top of this, a set of images will be compiled as a control group, to then be used for the generation of test images and for evaluation purposes.
After the optimization phase and model selection we close the project by conducting tests to validate the success of the process. Lastly, we will discuss the results achieved and compare them to the objectives, hypotheses and predictions formulated at the beginning of the project. We will also display the ways in which this project could be improved or expanded.

Key words

Deep Style, Convolutional Neural Network, Machine Learning, optimization, Deep Dream, Torch, style transfer, regression models.

Agradecimientos

Este trabajo ha sido posible gracias a Eduardo Garrido, quien siempre me ha ofrecido alternativas y soluciones cada vez que comenzaba a desviar la dirección del proyecto. No solo no restringió esta tendencia exploratoria sino que la fomentó, ayudándome a sacar el trabajo adelante. Siempre pendiente de que este trabajo cogía forma ha sido un factor muy importante en el desarrollo de esta memoria.
También agradecer a Alberto, Junior y David por su apoyo incondicional a lo largo de estos 4 años. Sin vosotros el camino hasta la cima probablemente hubiera sido igual de difícil, pero indiscutiblemente mucho menos entretenido.
Muchas gracias a todos aquellos profesores que me han enseñado con ilusión contagiosa por la materia, que han conseguido cultivar mi pasión por la programación y establecer mi vocación. Ha sido un largo proceso, pero después de 24 años estoy donde siempre soñé con encontrarme. Es cierto lo que dicen: Todos los caminos llevan a Roma. Lo que por fin entendí es que esto es cierto incluso cuando tú desconoces donde está.

Termino la Ingeniería,

Y esto acaba de empezar.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

Desde que el Internet fue creado en los años 60, el número de usuarios que hacen uso de la plataforma ha ido creciendo vertiginosamente. Este crecimiento tan acelerado no ha venido solo, con ello, la cantidad de datos que maneja cada usuario también ha sufrido un aumento semejante, gran parte de dichos datos siendo imágenes.
Compañías como **Google**, encargadas de indexar no solo páginas, sino imágenes, vídeos y otros tipos de archivo multimedia embebidos en ellas; a menudo se enfrentaban a un gran problema: archivos que no disponían de información, mas allá del nombre (a menudo no descriptivo), tamaño y tipo. Esto suponía una incapacidad de relacionarlos a otros archivos similares e incluso de acceder a ellos por métodos convencionales. Para abordar dicho problema, entre 2006 y 2011 el **Google Image Labeler** se hacía cargo de describir imágenes que carecían de información, invitando a usuarios jugar a describir imágenes aleatorias para poder brindar información más rica en detalles a otros usuarios, así ayudando a **Google** a tener una base de datos de imágenes mucho más efectiva, permitiendo acceso y relacionando imágenes.
Pero con más de 657.000 millones de imágenes subidas a la red anualmente[8], el etiquetado manual de imágenes se volvió rápidamente en una solución inviable, y de nuevo, una gran cantidad de imágenes volvían a quedar carentes de información para poder ser encontradas y relacionadas. Esto impulsó el desarrollo de una herramienta capaz de hacer frente a dicho problema a una mayor escala y de manera automatizada; Esta herramienta fue nombrada **Inception**, y fue desarrollada por **Google** en 2014 para el *Image-Net Large Scale Visual Recognition Challenge*[9]. Basada en una red neuronal convolucional, era capaz de detectar patrones que se podían corresponder con objetos, animales y personas en imágenes y etiquetar de manera acorde la imagen mediante lo que se llama pareidolia algorítmica. No se limitaba a la búsqueda de un único agente en la imagen, era capaz de discernir varios agentes, así como su posición en la foto e incluso combinación, como por ejemplo, una mujer con gafas.
La creación de este software tuvo un afortunado efecto secundario, al igual que podía detectar agentes en una imagen, también era capaz de introducirlos en ella utilizando de manera inversa la red una vez entrenada, así nació lo que se conoce como **Deep Dream**.
Dependiendo de arquitectura de la red que se utilice para realizar un proceso de modificación sobre la imagen de entrada existen varios tipos de variantes del **Deep Dream**. En este estudio nos enfocaremos en una variante en concreto, el **Deep Style**. Esta variante de la herramienta **Deep Dream** consiste en la aplicación de un estilo a una imagen, por lo tanto, se necesitan 2 elementos; una imagen de contenido y un estilo, para formar una de salida.
Se comienza tomando el estilo, se analiza y se obtiene información que lo define, reduciéndolo

a una combinación de tendencias (trazos, texturas, puntillismo, uso de líneas rectas exclusivamente), coloración (colores predominantes, distribución, saturación, contraste) y otros elementos como composición (distribución de los dos factores anteriores por toda la imagen, cantidad de detalle en las distintas partes de la imagen).
Tras sintetizar un modelo asociado al estilo, se intentan buscar en la imagen de contenido patrones similares a la imagen de estilo y se fuerzan a ser introducidos o amplificados en la foto a lo largo de 500 a 1000 iteraciones donde se va ganando claridad y legibilidad a medida que avanza el proceso. Esencialmente pidiendo al programa que exagere cualquier patrón conocido que vea en la imagen numerosas veces resultando en una aplicación total del estilo sobre la imagen de contenido.
En este estudio se pretende sacarle el máximo partido a **Deep Style** realizando numerosas pruebas y utilizando **Machine Learning** para elegir configuraciones de red que puedan asegurarnos resultados legibles y subjetivamente atractivos.

1.1. Descripción del proyecto

A continuación, se mencionará el objetivo de este estudio y la razón por la que se ha decidido llevar a cabo el proceso. También se explicarán de las restricciones existentes y que soluciones se proponen para solventar o mitigar los problemas que crean dichas restricciones. Por último, se expondrán las hipótesis formuladas previamente al desarrollo del proceso realizado en la sección *3.6-Desarrollo*.

1.1.1. Objetivos y Motivación

La utilidad de dicha aplicación del **Deep Dream** que vamos a desarrollar es puramente recreativa, pero existen variantes que aportan beneficios en la extracción de información de imágenes, reconstrucción y estimación de contenido como se comentará más adelante.
Deep Dream aún es una herramienta en un estado muy joven y con potencial para evolucionar de muchas maneras, el éxito o fracaso de este estudio podría contribuir al desarrollo de un modelo de aprendizaje para la optimización de distintas aplicaciones del algoritmo. Debido a la naturaleza del **Deep Style**, el cual vamos a tratar de optimizar, probablemente sea necesario adaptar o cambiar el funcionamiento del proceso de aprendizaje del algoritmo para que pueda ser transportada a otras aplicaciones de la herramienta **Deep Dream** con resultados aceptables.
Dicho esto, el proceso conlleva mucho tiempo (Puede tomar aproximadamente 2 horas obtener una imagen de 500x500 pixeles de tamaño en un ordenador que carezca de tarjeta gráfica) y no es fácil conocer *a priori* la calidad del resultado que nos brindará una configuración, puesto que estamos pidiendo a una herramienta que analice objetivamente una fotografía o una pieza de arte, y sustraiga de ambas información útil que permita combinar la red con la imagen dando un resultado que no solo sea discernible, sino subjetivamente interesante y agradable a su vez.
Este segundo objetivo no es posible cumplirlo de manera consistente puesto que no se puede juzgar objetivamente, por lo tanto, se medirá la calidad del producto con atributos objetivos, valorados subjetivamente. También se dará una valoración subjetiva de la calidad del resultado, juzgada de manera personal y lo mas imparcial y consistente posible, puesto que el propósito final de este proceso es obtener imágenes atractivas. Estas valoraciones serán explicadas mas extensamente en el apartado *3.5.2-Atributos de salida*.
Debido al exorbitante alcance de este objetivo se ha decidido limitarlo a únicamente fotos de retratos de personas, lo cual normaliza las entradas a un conjunto *cuasi* homogéneo a diferencia de un conjunto en el que se lidie con paisajes, fotos de ciudades, dibujos y retratos mezclados.
Tras definir el alcance de este estudio, para la mejora del algoritmo **Deep Style** se tratará

de encontrar patrones, tendencias y afinidades entre valores de atributos que producen buenos resultados para poder realizar conjeturas bien fundamentadas del posible resultado de ciertas configuraciones, o garantizar una alta probabilidad de un buen resultado con la utilización de ciertos parámetros. En estas conjeturas también tendremos en cuenta distintas características del retrato que utilizamos como imagen de contenido, las cuales describiremos en el apartado *3.5.1-Atributos de entrada*.
Con la información que se puede obtener de este estudio, en un futuro seria posible crear un sistema de recomendación de parámetros al usuario que prometa un resultado atractivo, o por otro lado que automáticamente aplique los parámetros que teóricamente puedan dar mejores resultados sin necesidad de que el usuario necesite intervenir. Esto podría suponer un ahorro de tiempo considerable y una posibilidad de personalización mayor sin sacrificar la calidad de la imagen de salida.

1.1.2. Restricciones

Como se ha mencionado en el apartado de objetivos y motivación, la principal restricción existente en este proyecto es la restricción temporal. Además, esta restricción temporal no solo surge por el proceso de generación de imágenes, sino también es consecuencia del método de evaluación, el cual no es automático y supone un tiempo añadido al que ya se utiliza para el proceso de generación de imágenes. Debido a esto se ha de ser cuidadoso definiendo el tamaño de las baterías de pruebas y el número de generaciones de imágenes ya que realizar cantidades excesivas de ejecuciones penalizará el progreso del estudio.
Por otro lado, existe la restricción de *hardware*. Las redes utilizadas en este tipo de algoritmos usan gran cantidad de recursos, la insuficiencia de estos produce el fallo de ejecución del programa. Esto significa que tener acceso a *hardware* potente es indispensable para realizar el estudio, particularmente la tarjeta gráfica. Soluciones alternativas a este problema son utilizar configuraciones menos optimas, que procesen imágenes de menor tamaño o usar equipos remotos disponibles para alquilar. Afortunadamente para este estudio se cuenta con una tarjeta gráfica de gama alta que puede realizar ejecuciones de buena calidad en un tiempo asequible.
También existen restricciones provenientes del *software* a utilizar. Debido a que es un *software* desarrollado por una persona con un fin no lucrativo, el programa no se robusto, existe documentación limitada y la capacidad de resolución de errores no es óptima. Esta clase de problemas pueden comprender desde incompatibilidades con *hardware* hasta incompatibilidad de *software*, como versiones de librerías, programas, *drivers* de la tarjeta gráfica e incluso sistemas operativos que pueden causar problemas a la hora de ejecutar. Por ello se intentará hacer uso de las mismas versiones usadas por el desarrollador del programa en la medida que sea posible.

1.1.3. Hipótesis

Antes del comienzo del estudio se han realizado una serie de conjeturas sobre el proceso que se define en el apartado *3.6-Desarrollo*. La primera hipótesis es que en el proceso de optimización de parámetros de este estudio se lleguen a valores similares a los propuestos como predeterminados por la implementación que se utilice.
También se teme que el *dateset* de información pueda resultar insuficiente para la clasificación y predicción de puntuaciones. Esto podría ser a causa de insuficientes atributos descriptivos de la entrada o la imposibilidad de definir atributos relevantes, tanto de entrada (como podrían ser métricas de imagen) como de salida (proceso evaluativo fundamentalmente errado). Otra causa podría ser el tamaño insuficiente de datos de entrenamiento del modelo de regresión. Al tratar

con datos altamente sensibles a un juicio subjetivo la falta de datos objetivos puede suponer la necesidad de más datos o más información.
También se predice que la calidad de las imágenes en ejecuciones iniciales del desarrollo resulte pésima, ya que serán con parámetros previos al proceso de afinamiento. No se puede saber con antelación el grado de mejora que se pueda conseguir a través la aplicación de procesos de afinamiento de valores de los parámetros.
Por último, es posible que la calidad de las imágenes de salida dependa excesivamente de la imagen de estilo utilizada. Esto también podría suponer un problema, ya que no existe un proceso definido para la selección de las imágenes de estilo.

2

Estado del arte

2.1. Introducción

Los dos elementos principales que componen este estudio, **Deep Dream** y **Machine Learning**, se encuentran en una fase de crecimiento vertiginoso, puesto que ambas cuentan con un gran potencial el cual ha sido reconocido y se está comenzando a explotar. Este alto interés, también se debe a la gigante versatilidad de ambos, que encuentran cabida en prácticamente cualquier ámbito en forma de una herramienta capaz de optimizar la ejecución de una tarea o predecir resultados con un mayor grado de precisión que humanos con ayuda de herramientas existentes.

2.2. Machine Learning

Cuando hablamos de **Machine Learning**, el propio concepto ya es muy amplio de por sí, pero el potencial de esta rama de la Inteligencia Artificial ya se ha visto desplegado en numerosas ocasiones. A continuación, daremos unos ejemplos en los que el **Machine Learning** ha sido fundamental para el desarrollo de software.
Por ejemplo, para los motores de búsqueda el **Machine Learning** ha aportado soluciones de reconocimiento de patrones, discernimiento de palabras valiosas en consultas, medición de importancia de distintos documentos en base a eventos o noticias de la actualidad, búsqueda de imágenes inversa, corrección de faltas de ortografía, selección de anuncios en base al usuario[10] , etc. También son un producto de **Machine Learning** aquellas herramientas que **Google** ofrece como respuesta a consultas especiales, como pueden ser operaciones matemáticas, valores en bolsa de empresas, ubicaciones o celebridades.
Continuando por otro uso similar, sus aplicaciones en Big Data consiguen transformar el manejo y procesamiento de inmensas cantidades de datos en una realidad, como es el caso de **Inception**, la herramienta ya nombrada en este estudio. Delegar esta tarea a una herramienta que mediante modelos y algoritmos puede rápidamente limpiar y sintetizar la información para posteriormente poder ser utilizada de manera más óptima ha permitido la rápida expansión del Internet.
También hace posible el funcionamiento de tecnologías como reconocimiento de voz, prevención de *ciberataques*, detección de correos *spam* y de fraudes con tarjeta de crédito.

Pero sus aplicaciones se extienden más allá del ámbito informático, filtrándose en campos como la medicina, donde se utiliza para la identificación de enfermedades y diagnósticos, desarrollo de tratamientos personalizados, investigación y descubrimiento de medicinas, detección y tratamiento de tejido canceroso a través de radiología y radioterapia, donde de nuevo, **Google** esta actualmente liderando la investigación con una herramienta denominada *DeepMind Health*. Como se puede comprobar, con **Machine Learning** basta con la implementación adecuada y suficiente información para desarrollar cualquier tarea y optimizarla, como veremos en este estudio, donde se propondrá una metodología y se proveerá al modelo de aprendizaje de información suficiente para aprender a predecir resultados de manera independiente.

2.3. Deep Dream

Por otro lado, en este estudio hacemos referencia a **Inception**, el cual es actualmente el algoritmo utilizado por **Google** para la clasificación y etiquetado automático de las imágenes que aparecen en la red, como se ha descrito anteriormente. Dentro de esta herramienta reside una red entrenada manualmente con una gran cantidad de imágenes. Esta red cuenta con numerosas capas de neuronas, las cuales detectan y clasifican las características de imágenes en diferentes niveles de abstracción. Comenzando con cosas simples como bordes u orientaciones; pasando por formas, curvas, polígonos y componentes básicos de objetos hasta llegar a la detección de elementos más complejos como personas, edificios, objetos, etc.
Tras el costoso entrenamiento de esta red tan compleja, se puede iniciar la fase de explotación. El proceso de categorizar y etiquetar imágenes es sustancialmente más rápido que el de entrenamiento de la red, permitiendo mantener el ritmo del crecimiento de la cantidad de imágenes subidas a diario en la red.
Desde su creación se ha pulido la capacidad de etiquetado y con ello ha aparecido la posibilidad de llevar a cabo un proceso más complejo. Describir el contenido de la foto. Una tarea que parece similar pero no resulta nada trivial para una red neuronal. Describir una imagen implica no solo reconocer los distintos agentes de una foto sino identificar qué relación existe entre ellos y formular oraciones en lenguaje natural que represente estas relaciones adecuadamente.

Figura 2.1: Ejemplo de oraciones generadas automáticamente por una red neuronal[1].

Pero su utilidad no se queda en solo la clasificación y descripción de estas imágenes, también se puede utilizar para entrenarse a si mismo mediante realimentación[11], introduciendo como datos imágenes *soñadas* mediante **Deep Dream**, es decir, imágenes modificadas introduciendo forzosamente agentes en ella siguiendo la metodología que describiremos a continuación.
Para convertir la herramienta de **Inception** en **Deep Dream** se le da la vuelta al proceso. En vez de entrenar una red neuronal con imágenes y utilizarla para reconocer y etiquetar agentes existentes en la imagen, utilizamos una red ya entrenada para forzar la búsqueda de patrones

conocidos e inserción de agentes en una imagen, procesándola por las denominadas octavas. Estas octavas son, para simplificar, la escala a la que se procesa la imagen y como resultado, el tamaño del agente que se fuerza a introducir. Esto significa que cuando la octava es baja (en conjunción con un número alto de iteraciones) los agentes introducidos son de menor tamaño, mientras que en octavas altas con pocas iteraciones comienzan a ser de mayor tamaño[12]. *A priori* se podría creer que octavas altas que producen agentes de mayor tamaño podrían resultar mejores, pero estas también traen consigo la generación de mucho ruido en el agente que introducen, en contraste con agentes más pequeños, donde el ruido se camufla por la menor resolución a la que han sido introducidos. La solución óptima y equilibrada es introducir agentes a lo largo de numerosas octavas. Para ello se hace uso de funciones que indican la densidad de agentes introducidos en las distintas octavas. Cada función da como resultado una distinta distribución de estas figuras, dependiendo si se eligen distribuciones lineales, binomiales o exponenciales[13].
A parte de la combinación de octavas e iteraciones también existen otros parámetros como la escala de octava (o escala de marco) que indica la ampliación que se produce de una escala a la siguiente, o la estructura de la red neuronal que, dependiendo de la arquitectura, numero de capas, tipos de capas y el orden pueden dar resultados muy dispares incluso compartiendo los demás parámetros.
A pesar de ser inicialmente una herramienta recreativa, la semejanza de las imágenes *soñadas* por el algoritmo con las alucinaciones inducidas por el *LSD* o la *psilocibina* indica que el funcionamiento de esta red neuronal podría ser muy parecida a ciertas capas de la corteza visual[14] lo que podría indicar posibles avances en campos biológicos y biotecnológicos, como por ejemplo el desarrollo de lentillas o prótesis oculares de realidad aumentada.
Encima de todo esto, la vistosidad de las imágenes que genera esta herramienta produjo un gran *bum* en popularidad cuando el código fuente fue publicado por **Google**[15] en 2015, permitiendo a cualquier persona experimentar con esta herramienta utilizando modelos entrenados proporcionados por el *framework* de *Caffe*[16]. Tal fue el impulso que incluso fue utilizado en un videoclip de *Foster The People - Doing It for the Money*, donde se utilizó la herramienta para modificar numerosos fotogramas del videoclip.

2.3.1. *Super Resolución. Upscaling* de imágenes

Dependiendo del *dataset* usado para el entrenamiento y de la arquitectura usada en la red neuronal también podemos conseguir que la red realice *Upscaling*. *Upscaling* o *Super Resolution* es la recuperación de una posible representación de alta calidad a partir de una imagen de baja calidad tras aumentar su escala[3]. Esta técnica permite, a través de una red neuronal *SRResNet*[17] y aplicando un algoritmo recursivo de análisis de los píxeles, suponer el contenido de las zonas entre píxeles en base a conocimiento adquirido anteriormente. Una técnica extremadamente útil, que aporta resultados creíbles, pero en algunos casos incorrectos. Unos ejemplos en los que la aplicación de la técnica revela información de forma fidedigna es en figuras geométricas, formas simples y líneas curvas o rectas, ya sean sin contexto o sobre animales, personas u objetos.

Figura 2.2: *Upscaling* resultado de distintos modelos. Arriba Izquierda - Interpolación bicúbica, Arriba Derecha - SRCNN, Abajo Izquierda - *Perceptual Loss*, Abajo Derecha - SRResNet[2].

Por otro lado, cuando procedemos a realizar una reconstrucción de rasgos más complejos, como pueden ser caras en muy baja resolución, *Upscaling* da resultados que a veces distan de la realidad, aunque puedan parecer plausibles. Esto es debido a que la reconstrucción se realiza con rasgos faciales aprendidos por la red neuronal, los cuales suelen ser refinados al normalizar gran cantidad de caras.

Figura 2.3: Reconstrucción de caras a través de una red neuronal[3].

La evolución de esta técnica seria muy importante en ámbitos de criminología e investigación forense, pudiendo hacer realidad escenas de series de crímenes donde es posible aumentar la resolución de imágenes de manera indiscriminada sin rendimientos decrecientes.

2.3.2. *Colorización de Imágenes*

También es posible entrenar una red neuronal capaz de aplicar color adecuadamente a imágenes en blanco y negro, localizando e identificando agentes en la fotografía y conociendo su habitual color al haber sido entrenada con anterioridad con un *dataset* suficientemente completo puede añadir matices a las distintas partes de una foto dándonos como resultado una increíble restauración de imágenes como se pueden ver a continuación.

Figura 2.4: Reconstrucción de color en imágenes a través de una red neuronal[4].

2.3.3. Traducción Imagen-Imagen

Otra manera de utilizar redes neuronales entrenadas es la traducción imagen-imagen, en la que podemos realizar un dibujo o un esquema y forzar a una red entrenada de forma muy específica a que represente siguiendo la estructura del dibujo los agentes con los que ha sido entrenado, como en este caso, en el que podemos hacer el esquema de una fachada con distintos colores representando puertas, ventanas, balcones, etc. La red se encarga de construir una imagen introduciendo los agentes especificados en las zonas designadas mostrando como resultado una fachada completamente artificial construida a partir de las imágenes con las que hemos entrenado el modelo.

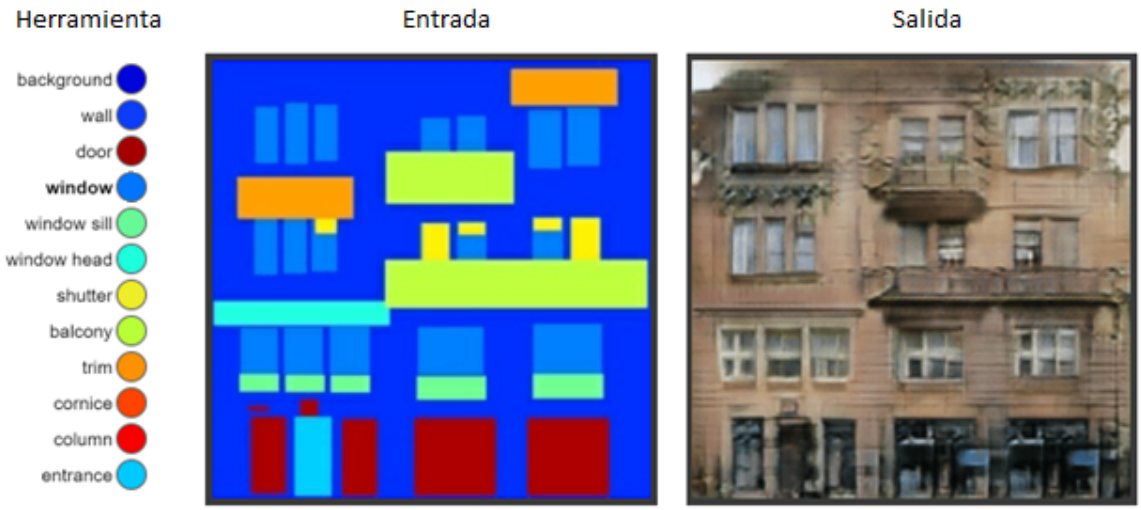


Figura 2.5: Creación de una fachada artificial a partir de un esquema con código de colores[5].

2.3.4. Reconocimiento de Texto

También es posible leer el texto que se encuentra en una imagen, si entrenamos la red con numerosas imágenes de letras en distintas rotaciones y con distintos grados de claridad podemos obtener una red neuronal capaz de leer texto en imágenes a pesar de la perspectiva, color, tamaño y desgaste o ruido de este. Esta aplicación está altamente desarrollada y es extremadamente útil para herramientas como **Google Translate**, o lectura de matrículas de coches que cometen infracciones.

Figura 2.6: Traducción en tiempo real de texto encontrado en imágenes por la herramienta de **Google Translate**[6].

3

Sistema, diseño y desarrollo

Para el desarrollo de este estudio se va a utilizar una implementación del algoritmo de **Deep Style** de entre las numerosas versiones que hay disponibles en la red. Todas estas implementaciones son el desarrollo del algoritmo descrito en el artículo de investigación de *Gatys et al* publicado en 2015 titulado *A Neural Algorithm of Artistic Style*[18], el cual tomo el reciente crecimiento del interés y uso de redes neuronales en tareas de procesamiento y extracción de información para crear un sistema de aplicación artística de un estilo a una imagen, ambos arbitrarios, resultando a su vez en avances en la investigación de la percepción visual humana.

3.1. Deep Style. Descripción del algoritmo

Como se ha descrito anteriormente, la herramienta **Deep Style** hace uso de 2 entradas. La primera es la imagen de contenido, la cual se utiliza como base para aplicar el estilo extraido de la segunda entrada, la entrada de estilo. Esta entrada, dependiendo de la implementación puede ser una imagen, o un archivo que contiene un modelo entrenado. Cada uno de estos tiene pros y contras, dependiendo de como querramos utilizar la herramienta.
Por un lado, si utilizamos una implementación que recibe una imagen de estilo, el programa otorga al usuario un mayor control sobre la salida. No solo en términos de selección de la imagen de estilo, que por si sola ya influye inmensamente en la salida, sino también en términos de ajuste y afinado. Cuando se permite al usuario aplicar el estilo de una imagen, también se le confiere la capacidad de controlar parámetros como la tasa de aprendizaje, la escala de la imagen de estilo, el color de la salida y los pesos de las variables de contenido, estilo, variación total. Todas estas variables están descritas con mayor nivel de detalle en la sección *3.3.2-Parámetros de la aplicación y sus efectos en la imagen de salida*.
Sin embargo, este mayor control sobre la generación tiene un inconveniente, obliga a que el proceso de la creación de imagen de salida cuente ahora con 2 pasos. Primero la extracción del estilo para entrenar la red neuronal con los parámetros personalizados, y segundo la aplicación de la configuración del estilo en la imagen de contenido. Esto hace el proceso significativamente más lento, precisamente porque donde esta el coste computacional es en el entrenamiento de la red.
Por eso se desarrolló un método para hacer frente a este problema, el *Instance Normalization*

(Normalización de Instancias). Este proceso entrena previamente el estilo con un *dataset* de imágenes variado (generalmente, el *dataset* de *Microsoft COCO*[19] de un tamaño aproximado de 20GB) que contiene fotografías de personas, animales y objetos en todo tipo de contextos, así como un subconjunto de imágenes para realizar una validación. El estilo se entrena con este inmenso *dataset* a lo largo de un gran número de iteraciones, generalmente entre 20.000 y 60.000, aprendiendo como aplicar el estilo en una región dependiendo de lo que este contenido en ella, generando lo que denominaremos un modelo. Esto hace que la herramienta solo tiene que observar y aplicar el estilo de una manera que ha sido predeterminada en la fase de creación del modelo. Con ello conseguimos una mejora en velocidad de creación de imágenes de salida importante, generando imágenes de 50 a 100 veces más rápido que por el primer método y con capacidad de generación de imágenes de mayor tamaño al no tener tanta restricción de memoria al haber hecho el trabajo pesado anteriormente. Desafortunadamente este proceso tiene una gran inconveniencia. El proceso de normalización de instancias es extremadamente lento, y hace uso de gran cantidad de memoria. Tras realizar pruebas con una tarjeta gráfica *NVIDIA GeForce GTX 1080*, una tarjeta gráfica de gama alta, el proceso tomó aproximadamente 5 horas. Siendo este el mejor de los casos (puesto que, si el proceso se realizase sin aceleración de *GPU* y utilizando el procesador en su lugar, podría tomar meses de continua ejecución), esta opción solo se habría de considerar si ya se cuenta con el modelo entrenado o si la cantidad de imágenes que se van a proceder a generar utilizando el modelo amortizan este costo de tiempo adicional (alrededor de 200 o 300 como mínimo).
Debido a que el objetivo de este estudio es optimizar el algoritmo para brindar mayor capacidad de personalización al usuario, la primera implementación resulta la opción más sensata, puesto que generar durante 4 horas un modelo del que no se pueda validar su utilidad hasta posteriormente no resulta eficiente.

3.2. Otras versiones de Deep Style. Implementaciones incompatibles

Tras haber mencionado los dos tipos principales de **Deep Style** procedemos a presentar distintas implementaciones de **Deep Style** que por varias razones no son compatibles (en distintos grados) con los objetivos que se buscan cumplir en este estudio.

3.2.1. Fast Neural Style. Implementación de Justin Johnson

Esta implementación[20] es una de las implementaciones más completas que se ha podido encontrar. No solo nos proporciona la generación de imágenes a través de imágenes de estilo y modelos, sino que nos otorga las herramientas necesarias para la aplicación del estilo a salida de vídeo de una cámara web en tiempo real. También contiene el código necesario para la generación de archivos de *dataset* (archivos *HDF5*) compatibles con el código de generación de modelos de estilo (archivos *.t7*).
Pero a pesar de poder generar imágenes usando como entrada una imagen de estilo, los parámetros de la aplicación no nos ofrecen demasiada flexibilidad y control en la generación de imágenes. Esto es debido a que los parámetros de personalización de la aplicación de estilo se encuentran a muy bajo nivel, modificando comportamiento a nivel de capa dentro de la red convolucional del programa. Este nivel de complejidad supondría una cantidad inmensa de pruebas y quedaría fuera del alcance de este estudio.

3.2.2. Fast Neural Style. Implementación de Pytorch

Esta implementación en Python[21], a parte de ya no ser mantenido por el desarrollador, solo nos proporciona la generación de imágenes utilizando modelos de estilo y no define muchos parámetros personalizables para la generación del modelo. Al ser un programa muy básico se ha tenido que descartar esta implementación para el estudio.

3.2.3. Fast Style Transfer. Implementación de Logan Engstrom

Una implementación basada en la implementación de Justin Johnson utilizando [22]. Al igual que la anterior, el principal problema de esta implementación es que solo nos posibilita la generación de imágenes a partir de modelos de estilo, además de no aportar suficientes parámetros para su personalización.

3.2.4. Deep Photo Style Transfer. Implementación de Fujun Luan

Después tenemos una implementación de **Deep Style** especializada en la transferencia de colores, en vez de estilo[23]. Para ello trabaja de forma paralela con la imagen de contenido y de estilo realizando cálculos y aplicando el estilo por zonas. Haciendo uso de una técnica denominada *segmentación semántica* el programa separa regiones de las imágenes de contenido y estilo en lo que se denominan canales. Estos canales son relacionados 1 a 1 entre imagen y contenido. Con esta relación se utiliza el proceso de **Deep Style** para aplicar de manera controlada el estilo al contenido de cada canal. Este proceso hace del *Deep Photo Style Transfer* una herramienta especializada, y como es propio de las herramientas especializadas, implica que existen restricciones.
En este caso, la principal restricción es la similitud contextual que ha de haber entre ambas fotos. Esto es un factor muy importante para el correcto funcionamiento del programa. También existe el problema de la necesidad de preprocesamiento de las imágenes para segmentarlas antes de ser utilizadas por el programa, y por ultimo, el resultado mediocre al procesar fotos, puesto que no aplica cambios vistosos.
La herramienta en si es una brillante derivación del **Deep Style**, pero no cuenta con la funcionalidad necesaria para el desarrollo de una herramienta como la que se ha marcado como objetivo.

3.3. Neural Style. Implementación de Justin Johnson

Tras haber descartado todas las demás opciones, se ha optado por utilizar la primera versión de la implementación utilizando *Torch* de Justin Johnson, la cual utiliza imágenes de estilo como entrada al igual que la versión actual, pero cuenta con más parámetros que añaden flexibilidad y personalización.
En este apartado explicaremos en detalle el funcionamiento de la aplicación y abarcaremos todos los parámetros que se van a explorar en este estudio.

3.3.1. Diseño de la implementación

Para comprender la implementación de Justin Johnson del **Deep Style** debemos entender el funcionamiento de las denominadas redes neuronales convolucionales. Las redes neuronales convolucionales (**CNN**, *Convolutional Neural Network*) son redes neuronales multicapa inspiradas por el cerebro y su capacidad de percibir imágenes, por ello son, con diferencia, las redes más poderosas para el procesamiento de imágenes. A diferencia de otros tipos de redes, las **CNN** contienen capas en forma de matrices tridimensionales organizadas en secciones de dos dimensiones que hacen el papel de unidades computacionales que procesan la información de manera jerárquica propagándola a la siguiente capa. Además, las neuronas de cada capa no se conectan a todas las de la siguiente, solo a pequeñas regiones de esta. Por último, está la capa de salida, la cual se compone de un vector de probabilidades.
Las **CNNs** se compone de dos procesos: La extracción de características, llevada a cabo utilizando numerosas capas que a su vez se componen de múltiples filtros que utilizan convoluciones y procesos de *pooling* o agrupamiento para extraer características específicas de la imagen; y el proceso de clasificación, donde varias capas clasificaran la entrada a partir de las distintas características que reciben en los *feature maps* de entrada. Estas asignarán probabilidades de que el objeto que aparece sea reconocido como distintas cosas (Perro, gato, persona, niño, etc.).
Al principio, en el proceso de extracción de características el componente primario son las convoluciones. Una convolución se refiere a la combinación matemática de 2 funciones para producir una tercera, fusionando así dos sets de información. En el caso de los **CNNs** la convolución se realiza con la imagen de entrada y un filtro o *kernel* para obtener lo que se denominan *feature maps* o matrices de características[24][25].
El proceso de convolución se realiza deslizando sobre la matriz de valores (en el caso de una imagen, píxeles) una matriz filtro de menor tamaño y en cada posición realizar una multiplicación de matrices y sumar posteriormente cada elemento para producir un valor por posición del filtro sobre la imagen de entrada. En la siguiente imagen podemos ver el resultado de aplicar un filtro 3x3 **K** arbitrario sobre una entrada **I**.

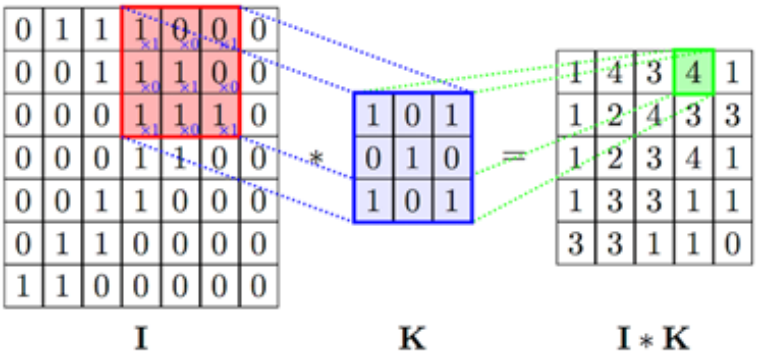


Figura 3.1: Proceso de aplicación de un filtro a una matriz de entrada[7].

Como se puede ver, la matriz de salida es más pequeña que la de entrada. Si se aplicase este proceso numerosas veces la matriz resultante seria diminuta comparada con la de entrada. Para resolver esto se pueden añadir celdas alrededor de la matriz de entrada a modo de *padding* o margen. Estas celdas de margen se suelen rellenar con ceros.

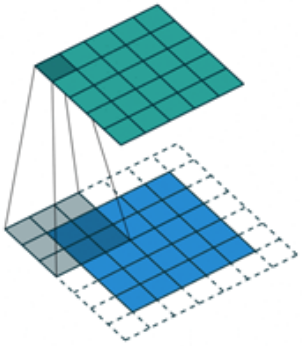


Figura 3.2: Matriz de entrada (azul) con celdas blancas haciendo la funcion de *padding* para obtener una matriz salida (verde) del mismo tamaño.

La tarea de convolución se realiza un número generoso de veces, cada una con un filtro distinto que obtiene una matriz resultado diferente, obteniendo así numerosos conjuntos de filtros que producen una variedad de características que van componiendo los *feature maps* de cada capa.
El proceso de convolución suele venir acompañado por la aplicación de una función de activación, para obtener una salida no lineal. En redes **CNN**, esta función de activación suele ser una función *ReLU* (*Rectified Linear Units*), la cual es una función simplificada que resulta en el mismo valor que la entrada, excepto si esta es negativa, en cuyo caso la salida será 0. Es necesaria una función lo menos compleja posible porque el número de veces que se aplica esta función en una red **CNN** comparado con otras redes es mucho mayor, y el coste computacional rápidamente se dispara.
Tras repetir los pasos de convolución-*ReLU* una o más veces se suele añadir una capa de *pooling*, la cual realiza un *downsampling* de la entrada. Este proceso es, esencialmente, reducir la resolución de la imagen manteniendo información importante de la entrada original. Esto se realiza para disminuir el coste computacional, aumentar la diversidad de filtros y controlar el sobreajuste. En la implementación de Justin Johnson que utilizaremos existen 2 tipos de *pooling*. El *max pooling* y el *average pooling*.

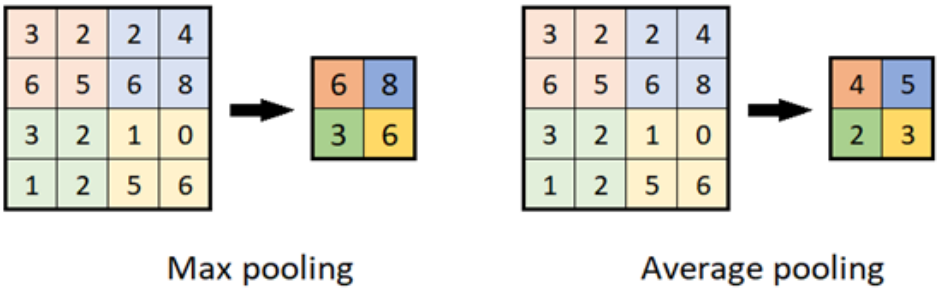


Figura 3.3: Proceso de *pooling* con filtros 2x2 haciendo saltos de 2 unidades.

Tras realizar este proceso de 3 pasos un número arbitrario de veces, comienza la fase de clasificación. Esta etapa se traduce en una serie de capas completamente conexas, pero esta vez, siendo capas son unidimensionales. Para ello tenemos que transformar nuestra salida de la etapa de extracción tridimensional en un vector unidimensional mediante la concatenación de cada fila y columna hasta formar un vector masivo que pueda aceptar la entrada del proceso de clasificación. Después de convertir la entrada, el proceso de clasificación es exactamente el mismo que el de una red neuronal multicapa convencional. Con esto, un ejemplo típico de la estructura de un **CNN** sería la siguiente:

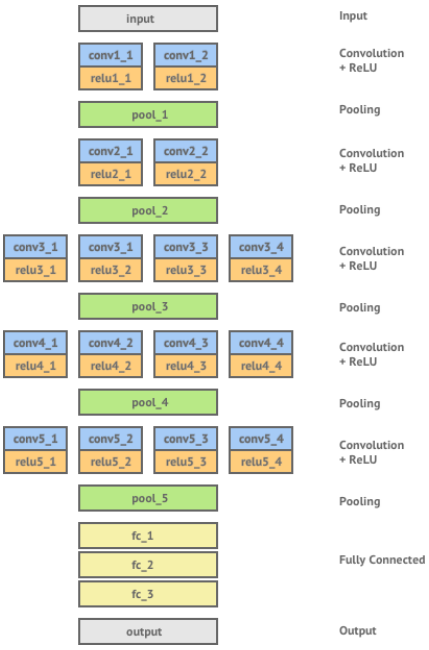


Figura 3.4: Estructura interna del VGG19 (*Visual Geometry Group* de 19 capas), un **CNN** comunmente usado en **Deep Style** .

Tras explicar la arquitectura de la red **CNN**, ahora pasamos a describir como es el proceso de aplicación del estilo a la imagen.
Para empezar, la idea de *aplicar el estilo a la imagen* resulta parcialmente errónea, puesto que ni la instancia de la imagen de contenido ni la de estilo son la base de la imagen de salida. En realidad, existen tres imágenes: La de contenido, la de estilo, y la de salida. Esta última puede estar inicializada de diferentes maneras, como podría ser ruido digital, una instancia distinta de la imagen de contenido, etc. Con estas 3 imágenes lo que realizamos son diferentes comparaciones para ver que grados de similitud existen de contenido-salida y estilo-salida, intentando minimizarlas simultáneamente. Aquí surge la necesidad de cuantificar el grado de similitud, y se realiza con las métricas de *Content Loss* (Pérdida de contenido) y *Style Loss* (Pérdida de estilo). Estos dos atributos representan la cantidad de información perdida en la imagen de salida respecto de las otras dos.
Para el *Content Loss*, por cada iteración cogemos la imagen de salida y la optimizamos iterativamente utilizando los *feature maps* de capas selectas de la imagen de contenido. La definición de capas específicas para la reconstrucción de la imagen es necesaria debido a que la imagen de salida ha de estar compuesta de características tanto de la imagen de contenido como la de estilo. Si utilizásemos todas las capas para la reconstrucción estaríamos efectivamente copiando la foto por fuerza bruta, y con resultados no ideales. Una vez conocidas las capas con las que se realiza la reconstrucción, se define como *Content Loss* de una capa **l** el error cuadrático medio entre la matriz de características **F** de la imagen de contenido **C** y la matriz de características **P** de la imagen de salida **Y**:

$$\ell_{content} = \frac{1}{2}\sum(F_{ij}^{l} - P_{ij}^{l})^2 \tag{3.1}$$

Cuando el *Content Loss* se minimiza, significa que la lista de características de la imagen de salida en una capa determinada es muy similar a la lista de características de la imagen de contenido en esa misma capa. Dependiendo de lo profunda que sea la capa seleccionada, esto podrá transferir características como contornos y formas complejas a la imagen de salida.
Para el *Style Loss* realizaremos un proceso similar, pero en el caso del estilo queremos saber

que capas de estilo se activan simultáneamente y copiar este patrón de activación a la imagen de salida. En la implementación de Justin Johnson, la cual sigue el algoritmo definido en la publicación de *Gatys et al*[18], se realiza el cálculo de la matriz de Gram **G** de la capa **l** a partir de la matriz de características de la capa **l** de la imagen de estilo **F** de la siguiente manera:

$$G_{ij}^{l} = \sum_{k} (F_{ik}^{l} - F_{jk}^{l}) \tag{3.2}$$

Con esta matriz de Gram ya podemos aplicar la misma función que utilizamos para el *Content Loss* sustituyendo la matriz de *features* por la recién calculada matriz de Gram de las imágenes de estilo y salida **G** y **A**.

$$\ell_{style} = \frac{1}{2} \sum (G_{ij}^{l} - A_{ij}^{l})^2 \tag{3.3}$$

Al igual que ocurre con la imagen de contenido y el *Content Loss* si obtenemos matrices de Gram similares para la imagen de estilo y la imagen de salida podríamos anticipar que ambas imágenes tendrán un estilo similar, pero no necesariamente el mismo contenido. Por lo tanto, es común elegir capas superficiales para realizar la reconstrucción del contenido, pues estas capas suelen capturar mejor las texturas en contraposición a elementos de más alto nivel, pero para buenos resultados, generalmente se utiliza una combinación de capas superficiales y profundas para la reconstrucción del estilo, puesto que a menudo el estilo no se compone solo de texturas sino de elementos de gran tamaño distribuidos por la imagen.
Con estos dos factores, vamos modificando la imagen de salida iterativamente, poco a poco forzándola a parecerse a la imagen de contenido y la de estilo simultáneamente, asemejándose a cada una en las respectivas capas definidas, para así obtener una tercera métrica, el *Total Loss* (Pérdida total), definida como una suma ponderada de las dos anteriores:

$$\ell_{total} = \alpha \ell_{content} + \beta \ell_{style} \tag{3.4}$$

Dependiendo de los valores de α y β, pesos de contenido y estilo, la imagen de salida tendrá una aplicación más o menos sutil del estilo y, como consecuencia, será más o menos reconocible la imagen de contenido usada como entrada. A medida que se vaya iterando, esta pérdida irá disminuyendo hasta converger a un valor, a partir del cual la mejora por iteración es inapreciable, haciendo aconsejable la finalización del proceso.
Cabe añadir que existe también un proceso, no mencionado en la publicación de *Gatys et al*[18], pero si implementado por Justin Johnson, para lidiar con el ruido que genera este proceso. Este ruido se cuantifica con el *Total Variation Loss*, el cual calcula la variación entre los píxeles y su vecindad inmediata para minimizar los denominados *digital artifacts*. La utilización de este factor en una medida controlada ayuda a mitigar este problema, pero la asignación de un peso desmedido para este factor resulta en un difuminado excesivo de los colores en la imagen de salida.

3.3.2. Parámetros de la aplicación y sus efectos en la imagen de salida

En este apartado hablaremos de los distintos parámetros de la aplicación, realizando una descripción detallada de su función y también veremos representados los efectos de los parámetros generando imágenes variando de manera aislada cada uno. Por cada parámetro, en caso de ser continuo se probaran valores progresivamente más grandes y más pequeños partiendo del valor predeterminado recomendado. Para los atributos categóricos, se tomara un conjunto de parámetros que se parezca al recomendado, en caso de que el conjunto de valores que puede tomar el parámetro es suficientemente pequeño, se tomaran todos los valores.

Content Weight

También definido como α, es el coeficiente por el que se multiplica el *Content Loss* para el cálculo del *Total Loss*, permitiendo controlar como de reconocible será el contenido en la imagen de salida. El valor por defecto es $5 \cdot 10^0$ y se han considerado $5 \cdot 10^{-2}$, $1 \cdot 10^{-1}$, $5 \cdot 10^{-1}$, $1 \cdot 10^0$, $5 \cdot 10^0$, $1 \cdot 10^1$, $5 \cdot 10^1$, $1 \cdot 10^2$ y $5 \cdot 10^2$.

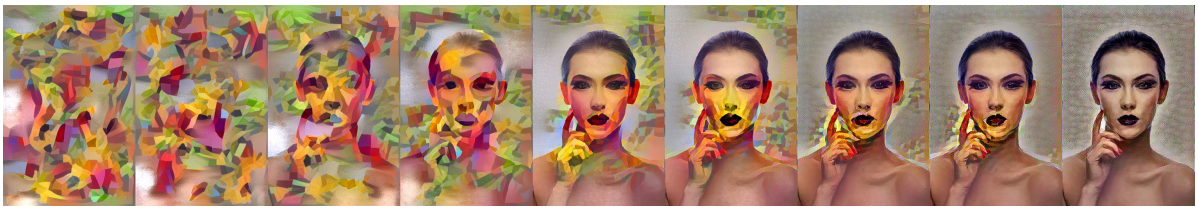


Figura 3.5: Consecuencias de los distintos valores del *content weight*, de menor a mayor valor.

Style Weight

El coeficiente β, el cual es el complementario del *Content Weight* que multiplica el *Style Loss* para el cálculo del *Total Loss*, dictando el grado de presencia del estilo en la imagen de salida. El valor por defecto es $1 \cdot 10^2$ y se han considerado $1 \cdot 10^0$, $5 \cdot 10^0$, $1 \cdot 10^1$, $5 \cdot 10^1$, $1 \cdot 10^2$, $5 \cdot 10^2$, $1 \cdot 10^3$, $5 \cdot 10^3$ y $1 \cdot 10^4$.

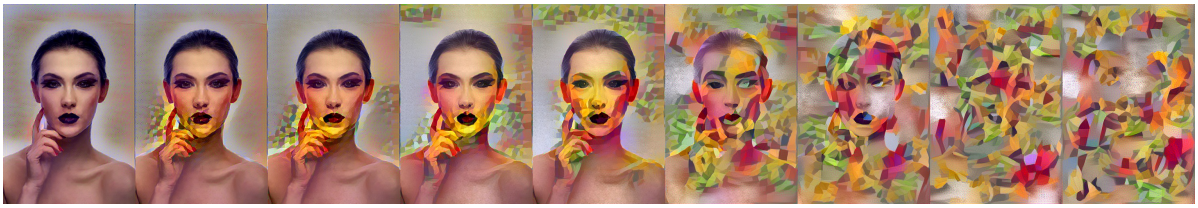


Figura 3.6: Consecuencias de los distintos valores del *style weight*, de menor a mayor valor.

TV Weight

Es el coeficiente que multiplica al termino de *Total Variation Loss* mencionado al final del apartado anterior, midiendo la variación entre cada píxel y su vecindad, es el encargado de realizar un proceso de suavizado en la imagen. El valor por defecto es $1 \cdot 10^{-3}$ y 0 indica que no se realiza suavizado, se han considerado 0, $1 \cdot 10^{-5}$, $5 \cdot 10^{-5}$, $1 \cdot 10^{-4}$, $5 \cdot 10^{-4}$, $1 \cdot 10^{-3}$, $1 \cdot 10^{-2}$, $5 \cdot 10^{-2}$ y $1 \cdot 10^{-1}$.

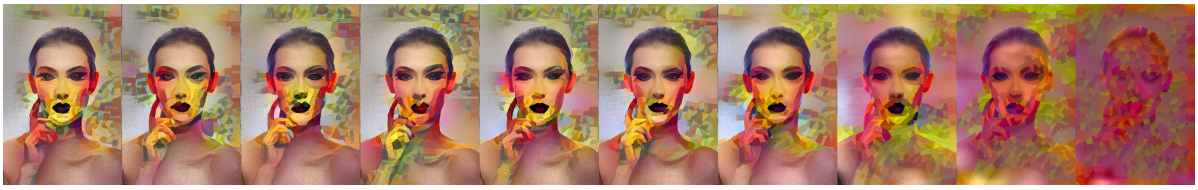


Figura 3.7: Consecuencias de los distintos valores del *tv weight*, de menor a mayor valor.

Learning rate

La tasa de aprendizaje es el coeficiente que indica la cantidad de la imagen de entrada que se introduce en cada iteración en la imagen de salida. Solo está presente con el algoritmo de optimización *ADAM*, el cual es el único que utilizamos para la generación de imágenes en este estudio. El valor por defecto es $1 \cdot 10^1$ y se han considerado $1 \cdot 10^{-2}$, $1 \cdot 10^{-1}$, $1 \cdot 10^0$, $5 \cdot 10^0$, $1 \cdot 10^1$, $5 \cdot 10^1$ y $1 \cdot 10^2$.

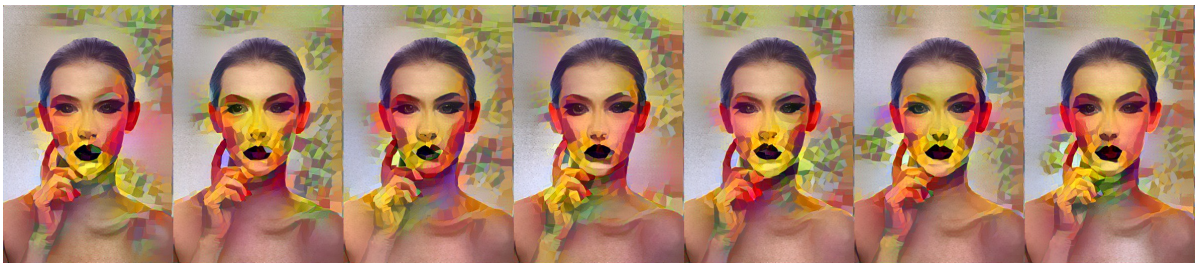


Figura 3.8: Consecuencias de los distintos valores del *learning rate*, de menor a mayor valor.

Content Layers

Son las capas que se utilizan para la reconstrucción de contenido en la imagen de salida separadas por comas, usando la notación del modelo *VGG19*. Para cada una de estas salidas se realiza el cálculo definido en *la ecuación de Content Loss*. Como se ha mencionado en la descripción de la implementación, las capas más profundas suelen ser las más recomendables para la reconstrucción de contenido. El valor por defecto es `relu4_2` y se han considerado los siguientes valores.

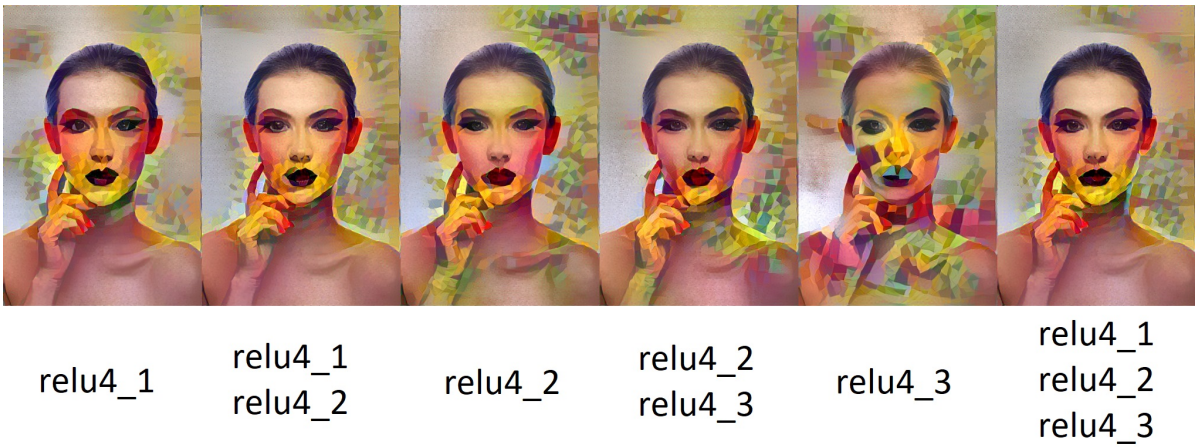


Figura 3.9: Consecuencias de los distintos valores del *content layers*, etiquetados debajo.

Style Layers

Al igual que tenemos la opción de elegir las capas de reconstrucción de contenido, podemos también elegir que capas utiliza para la reconstrucción del estilo con la notación *VGG19* separándolas por comas. En este parámetro es recomendable la combinación de capas superficiales y profundas para obtener las características en todos los niveles de abstracción del estilo. El valor por defecto es `relu1_1, relu2_1, relu3_1, relu4_1, relu5_1` y se han considerado los

siguientes valores.

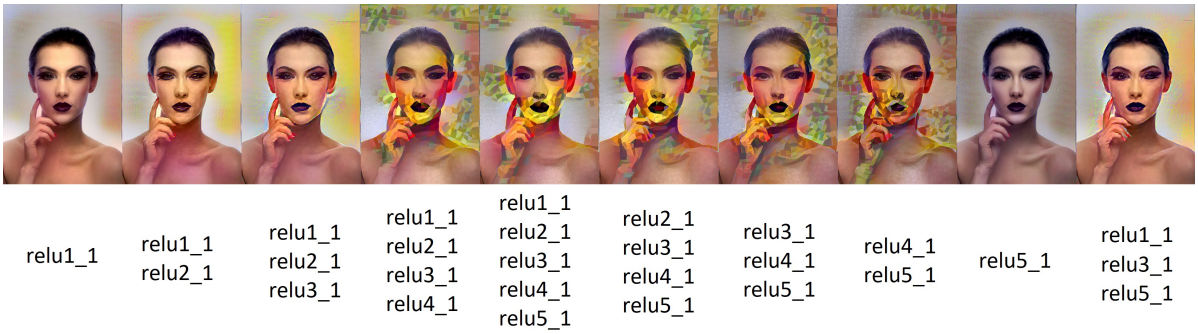


Figura 3.10: Consecuencias de los distintos valores del *style layers*, etiquetados debajo.

Original Colors

Flag que indica a la red si ha de mantener el matiz y colores de la imagen de contenido al aplicar los cambios de la imagen de estilo. No afecta a la reconstrucción del contenido. El valor por defecto es 1 (*true*) y se han considerado ambos casos.

Figura 3.11: Consecuencias de los distintos valores del *original colors*. A la izquierda el *flag* se encuentra activado, a la derecha se encuentra desactivado.

Pooling

Estrategia de *pooling* utilizada para realizar el proceso de *downsampling* de la imagen entre capas convolución-*ReLU*. Afecta a los *feature maps* obtenidos de las imágenes y como consecuencia, a la aplicación de estilo y contenido a la imagen de salida. El valor por defecto es *max* que indica *max pooling* y se han considerado *max pooling* y *average pooling*.

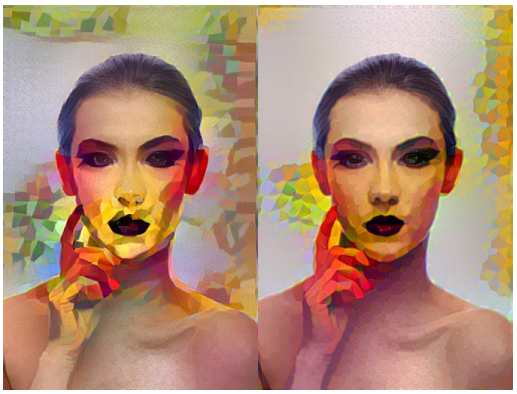


Figura 3.12: Consecuencias de los distintos valores del *pooling*. A la izquierda *max pooling*, a la derecha *average pooling*.

Style Scale

Indica el *upsampling* que se le aplica a la imagen de estilo. Esencialmente reescala la imagen de estilo para que en la aplicación se vea reflejado el cambio. Altos valores de este parámetro producen fallos en la ejecución por falta de memoria *VRAM* puesto que la resolución de la imagen aumenta, aunque sea de manera artificial. El valor por defecto es 1 y se han considerado 0,1, 0,2, 0,4, 0,6, 0,8, 1, 1,1 y 1,2.

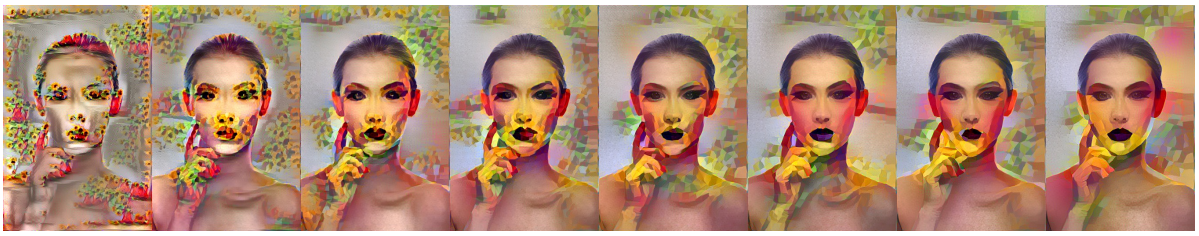


Figura 3.13: Consecuencias de los distintos valores del *Style Scale*, de menor a mayor valor.

A parte de estos parámetros existen otros que se han afinado de manera previa para prevenir fallos por insuficiente memoria, minimizar el tiempo de ejecución y ayudar con el proceso de seguimiento y depurado. Estos parámetros son los siguientes:
El tamaño de la imagen de salida, que viene definido por el número de píxeles que tendrá la dimensión más grande (manteniendo las proporciones de la imagen de contenido), el cual se ha fijado a 400 píxeles permitiendo un alto grado de holgura para valores altos de la escala de estilo y además agilizando el proceso de generación de imágenes.
El número de iteraciones, que indica cuantas veces se realizan las reconstrucciones de contenido y estilo en la imagen de salida antes de terminar la ejecución del programa. Para ello, previamente se han hecho algunas pruebas y se ha determinado que a partir de unas 600 iteraciones la mejora producida por cada iteración se ve tan reducida que es recomendable parar para disminuir el tiempo de ejecuciones masivas.
Además de estos dos parámetros se han utilizado los *flags* de `-gpu 0` (para indicar que se utilice la tarjeta gráfica para el proceso), `-init random` (para inicializar la imagen de salida con ruido aleatorio), `-print_iter 100` (para imprimir el *Content Loss* y el *Style Loss* cada 100 iteraciones) y `-save_iter 0` (para deshabilitar la exportación de imágenes intermedias).

3.4. Proceso de selección y presentación de datos

Para la optimización de esta implementación de **Deep Style** tenemos que contar con un conjunto de imágenes de contenido y otro de imágenes de estilo que servirán como grupo de control. Con estos dos conjuntos realizaremos las baterías de pruebas de manera controlada en lugar de ir seleccionando arbitrariamente imágenes nuevas en cada prueba que se realizase. En los siguientes apartados explicaremos que se ha tenido en cuenta en el proceso de selección de las imágenes para definir el conjunto de imágenes de entrada.

3.4.1. Imágenes de contenido

Como ha sido indicado en el apartado *1.1.1-Objetivos y Motivación* se ha reducido el alcance de este estudio a únicamente retratos, y fotografías de personas. La motivación detrás de la elección de este tipo de imágenes es la tendencia de usuarios a utilizar herramientas de transferencia de estilo para fotografías propias, a menudo con resultados mediocres. Centrando la optimización alrededor de este tipo de imágenes de entrada se espera desarrollar un modelo que permita recomendar una serie de parámetros de la aplicación al usuario que teóricamente puedan brindar resultados atractivos en base a las características de la fotografía que se quiere utilizar para el proceso de transferencia de estilo. Para crear un dataset lo más completo posible a la par que manejable (puesto que es necesaria la evaluación manual, la cual es notablemente más lenta y tediosa que cualquier tipo de clasificación automática) se han obtenido 102 retratos variados de www.pexels.com que contiene una gran base de datos de fotos de retratos, todos bajo una licencia *Creative Commons Zero* (*CC0*). Esta licencia dicta que las imágenes son para libre uso personal o comercial y que las imágenes se pueden modificar, copiar y distribuir, lo cual encaja perfectamente para la función que van a desempeñar en este estudio.

Figura 3.14: Algunas de las imágenes con las que se van a realizar las pruebas.

3.4.2. Imágenes de estilo

Puesto que las preferencias del estilo a aplicar son altamente subjetivas, las imágenes de estilo seleccionadas han sido mayormente arbitrarias, siguiendo criterios relajados basados en experiencias con ejecuciones previas al estudio. Uno de los criterios seguidos es la necesidad de que la imagen de estilo tenga una distribución homogénea del estilo, esto quiere decir, evitar vacíos en las imágenes de estilo, o cambios de estilo bruscos en secciones de la imagen para que la extracción de estilo resulte más equilibrada. Otro criterio usado es la carencia de contenido,

se han intentado seleccionar imágenes que carezcan de contexto, donde no se vean representados personas, animales u objetos, con el fin de ayudar en el proceso de abstracción del **CNN**. Por último, se ha tendido a seleccionar estilos que hagan uso de formas poligonales, las cuales resultan, subjetivamente, más atractivas al ser aplicadas sobre las formas orgánicas de una cara y cuerpo humano.

Figura 3.15: Algunos de los estilos que se van a transferir a los retratos.

3.5. *Dataset.* Atributos de entrada y Atributos de salida

Para formalizar los datos y poder utilizarlos para el entrenamiento de un modelo de clasificación es necesario transformar los datos en tuplas de un dataset. En nuestro caso los atributos de entrada del dataset representaran información de entrada del programa de transferencia de estilo y las salidas representaran la calidad del producto obtenido a partir de los atributos de entrada definidos. Para este estudio se van a tener en cuenta los siguientes parámetros.

3.5.1. Atributos de entrada

Este conjunto de atributos define toda la información necesaria sobre los datos de entrada de la aplicación. No solo se compone de los parámetros de entrada sino también de atributos asociados a la imagen de entrada definidos para controlar y cuantificar distintos aspectos de la imagen con el fin de aumentar la calidad del proceso de clasificación del modelo. Es por eso que se han definido 4 atributos asociados a la imagen de entrada, todos contenidos en el rango `[0-4]`.
Primero tenemos el atributo de perspectiva. Este define la frontalidad de la cara del sujeto. 0 indica u retrato frontal, 2 indica un retrato de tres cuartos y 4 indica que el sujeto está de perfil. Se utilizaran 3 y 1 en casos intermedios.

Figura 3.16: El valor de perspectiva de estas imágenes sería 0, 2 y 4, respectivamente.

Después sigue el atributo de aislamiento, el cual define la diferencia del nivel de detalle entre el sujeto y la escena, es decir, indica en que grado la persona es el centro de atención en la imagen. 4 indica que no existe ninguna clase de detalle en la escena, es decir, la persona esta frente a un fondo plano. 0 indica que la escena y el sujeto tienen el mismo grado de nitidez y detalle.

Figura 3.17: El valor de aislamiento de estas imágenes sería 4 y 1, respectivamente.

También tenemos en cuenta la cantidad de cuerpo. Este atributo indica la proporción entre cara y cuerpo del sujeto en la imagen. 0 nos indica que el encuadre deja dentro de la imagen solo la cara, pudiendo cortar la barbilla o la parte superior de la frente. 1 tendría el cuello dentro del encuadre. 4 es una foto hasta la cintura.

Figura 3.18: El valor de cantidad de cuerpo de estas imágenes sería 1,2 y 4, respectivamente.

Por último, también se tiene en consideración el contraste del sujeto. Con este atributo se pretende medir el contraste entre luces y sombras en la cara del sujeto, es decir, el grado de definición de los atributos faciales. Un 4 sería el caso extremo en el que apenas existen tonalidades intermedias, existiendo partes de luz y sombra bien definidas en la cara del sujeto. Un 0 serían colores planos sin profundidad.

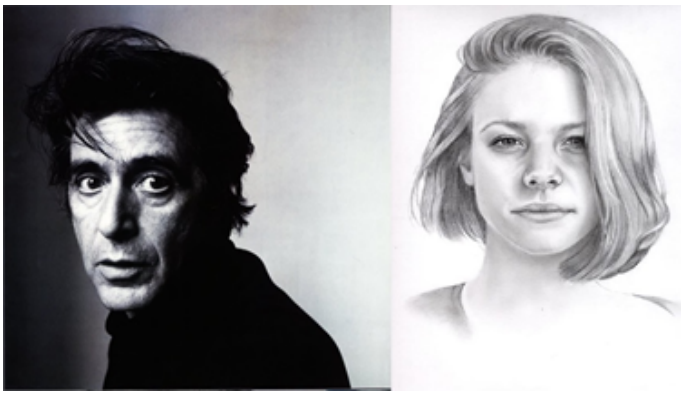


Figura 3.19: El valor de contraste de estas imágenes sería 4 y 1, respectivamente.

Sin embargo, para la imagen de estilo solo se ha utilizado como atributo la ruta al mismo por lo tanto el único control sobre estas imágenes se ha realizado en el proceso de selección mediante los criterios arbitrarios mencionados anteriormente. La adición de métricas en la imagen de estilo podría influir positivamente en el proceso de clasificación y optimización, pero desafortunadamente el coste temporal que supondría realizar pruebas con un conjunto de imágenes de estilo que proporcionasen una cobertura uniforme y suficiente del espacio de posibles valores imposibilita el desarrollo de una metodología de clasificación de imágenes de estilo.
Por último, en el conjunto de atributos de entrada almacenamos también los parámetros con los que se van a realizar combinaciones para generar distintas configuraciones de entrada del programa. Tanto la ruta de la imagen de estilo como los parámetros de la aplicación pertinentes serán tratados como atributos categóricos, aplicando sobre ellos la codificación *One-Hot*, para evitar que modelos de regresión como el *K-Nearest Neighbors* identifiquen valores parecidos como valores cercanos en el espacio. Con esto, la estructura de los atributos de entrada será la siguiente:

Retrato				Estilo	
perspectiva	aislamiento	cant. cuerpo	contraste	imagen estilo	...

Parámetros					
...	content weight	style weight	tv weight	learning rate	...

...	content layers	style layers	original colors	pooling	style scale

Cuadro 3.1: Estructura de la tabla de datos que contendrá el *dataset*.

3.5.2. Atributos de salida

Para completar el *dataset* se han definido 4 atributos de salida que en conjunto realizaran la función de calificar la salida obtenida por el programa utilizando los datos que se encuentran en los atributos de entrada. Los 4 atributos de salida, o clases, son números naturales en el rango `[0-10]` que tienen como objetivo evaluar distintos aspectos de la imagen de salida. A

continuación, se describen los 4 atributos de salida.
Primero tenemos el grado de discernimiento de la persona frente al fondo, indica cuanto destaca la silueta del sujeto en la imagen. Un 0 representa ruido aleatorio, 8 un grado de discernimiento igual al de la imagen original y se dejan 2 puntos de holgura hasta el 10 por si se consiguen resultados que resalten la silueta.
Después se evalúa el grado de discernimiento de los atributos faciales del sujeto de la imagen. Aplicando el mismo sistema de puntuación que en la métrica anterior evaluamos que los atributos faciales estén representados clara y fidedignamente en la imagen.
El tercer atributo se ha denominado captura de estilo, y es una métrica definida con el fin de evaluar si el programa ha conseguido aplicar de manera creíble la imagen de estilo que ha recibido. Esta evaluación tiene en cuenta si el estilo es reconocible, si no existen muchos patrones repetidos en la imagen y si la aplicación del estilo resulta en una imagen que no parece sintetizada a través de dos fuentes distintas.
Por último, tenemos la evaluación subjetiva de la imagen como producto. Aquí se realizará una evaluación calificando lo atractiva que resulta la imagen, posibles capturas de estilo o patrones que puedan resultar especiales y otros aspectos no considerados en las tres métricas anteriores. Esta evaluación sobre 10 tiene como punto de referencia el 5, que implicará que la aplicación del estilo no aporta nada artísticamente respecto de la imagen original.
Con esta metodología de evaluación se pretende calificar distintos aspectos de la imagen de forma objetiva con los tres primeros atributos, cada uno ponderado un 20 % para un peso total del 60 % de la nota y por otro lado dar una calificación subjetiva de la imagen como producto, que compondrá el 40 % de la nota. Con esto se busca obtener una nota representativa de la imagen juzgando legibilidad, calidad de las extracciones de contenido y estilo, y valor artístico.

3.6. Desarrollo

Tras haber definido todos los datos, recursos y metodologías que se van a utilizar para la creación y uso del *dataset* para el entrenamiento de modelos de predicción, podemos empezar a desarrollar el proceso de investigación.

3.6.1. Preprocesamiento

Una vez reunidas las imágenes de contenido y estilo en dos conjuntos se realiza una tarea de preprocesado para normalizar la nomenclatura de las imágenes. Para ello hemos reunido todas las imágenes de contenido en una carpeta denominada *input* y todas las imágenes de estilo las hemos introducido en una carpeta llamada *style*. Tras agrupar ambos conjuntos se ha desarrollado un pequeño programa en **Python** que se encarga de renombrar cada archivo de cada carpeta con la nomenclatura especificada. En este caso se han renombrado las imágenes de contenido como `portraitxxx.jpg` siendo `xxx` un índice de 0 a 101, y las imágenes de estilo como `styleyyy.jpg` siendo `yyy` un índice de 0 a 28. Tras realizar este proceso podemos rápidamente identificar cada imagen por su función (contenido o estilo) e índice y acceder a ella inequívocamente.

3.6.2. Definición de valores de atributos de los retratos

Tras normalizar los nombres de los archivos se puede comenzar con el proceso de definición de los valores de las 4 métricas descritas para las imágenes de contenido. Para agilizar el proceso se ha escrito un programa en **Java** que, por cada imagen en un directorio (en este caso,

el directorio *input*) recibe por teclado los valores de las 4 métricas, comprendidos entre 0 y 4, separados por espacios. Una vez introducidos y validados añade una línea a un fichero de datos que se comprende de 5 columnas. Nombre del archivo y el valor de las 4 métricas definidas.
Tras realizar el proceso de evaluación de las métricas de las imágenes de contenido contamos con la capacidad de representa una imagen en un *dataset* a través de un vector de 4 dimensiones. Con esto, el modelo podrá realizar predicciones basadas en imágenes con atributos parecidos cuando este recibe una imagen de contenido no existente en el conjunto de datos de entrenamiento, mientras se proporcione con la imagen dicho vector de atributos.

3.6.3. Optimización de parámetros previa. *Random Search Optimization*

Antes de comenzar a rellenar el *dataset* a ciegas con ejecuciones aleatorias, es importante reducir el espacio de búsqueda. Actualmente, con los valores de parámetros que pretendemos contemplar existen 4.609.920 posibles configuraciones por cada par de imagen de contenido e imagen de estilo. Con tal número de posibilidades realizar *dataset* de 300 a 500 datos no resulta representativo y obligaría al modelo a realizar muchas suposiciones para realizar predicciones ante entradas desconocidas, bajando notablemente el rendimiento del clasificador. Para tratar de reducir el espacio de búsqueda a uno más controlado se ha realizado una variación del *Random Search Optimization* que se adecúa a las necesidades del proceso que se desarrolla en este estudio.
El algoritmo de *Random Search Optimization* parte de un punto en el espacio de búsqueda, a partir del cual se generan puntos aleatorios dentro de una hiperesfera de radio previamente definido. Tras definir esta serie de puntos, estos se evalúan con una función de *fitness* y a partir de ellos se genera otro punto con una hiperesfera de radio menor según el resultado de la función de *fitness*. Este proceso se repite hasta que el valor del punto converja o se realice un número de iteraciones definido previamente. Esta metodología permite explorar espacios de búsqueda amplios donde se desconoce la consecuencia de la variación de los distintos parámetros, pero es incompatible con este estudio por las siguientes razones:
En este estudio, la evaluación de los nuevos puntos no se realiza automáticamente, es necesario que una persona sea la que realice el veredicto. Esto implicaría pausar la ejecución del algoritmo de optimización por cada punto evaluar. Este proceso se compone por la generación de la imagen, la cual implica una demora de entre 40 segundos y 2 minutos dependiendo de los parámetros utilizados, y posteriormente la evaluación manual. Si el proceso de optimización sufre interrupciones de alrededor de 2 minutos que, a su vez, obligan al evaluador a calificar una imagen cada intervalo de tiempo, la tarea crece en complejidad y coste temporal. Puesto que se estima que es necesaria la generación de alrededor de 2000 imágenes como mínimo para tener la suficiente información para realizar una optimización sensata, esto implica la necesidad de contar con un evaluador realizando aproximadamente una evaluación cada minuto o dos durante alrededor de 50 horas.
Esto nos introduce a la segunda incompatibilidad. La función de *fitness*. El coste temporal del proceso evaluativo utilizando las métricas definidas en *3.5.2-Atributos de salida* hace necesario desarrollar un proceso de evaluación simplificado el cual permita evaluar imágenes de manera más rápida, sin poner en riesgo la calidad de la información obtenida para la optimización.
Por ello se ha propuesto una variante del *Random Search Optimization*. Esta variante se compone de las siguientes etapas.
La primera etapa consiste en la generación de un número determinado de puntos aleatorios en el espacio de búsqueda definido. Por cada uno de estos puntos se producirá el comando necesario para una ejecución del programa para generar la imagen.
Tras definir estos puntos y obtener el comando se realiza una ejecución de los comandos masiva. Una vez generadas las imágenes se comienza la etapa de evaluación. En este caso se ha optado

por una metodología de clasificación simplificada a través de la cual se clasifica la imagen de salida como buena, mala o poco concluyente. Con este proceso simplificado de evaluación se puede realizar una clasificación de grandes lotes de imágenes de forma ágil, haciendo factible la evaluación de 2000 imágenes.
Por ultimo se realiza el proceso de descarte de valores de los parámetros. A partir de las imágenes clasificadas como buenas y malas se recoge la información de los valores más comúnmente involucrados en la generación de imágenes que se consideran atractivas y los valores que tienden a dar como resultado imágenes desagradables. Con este proceso obtenemos una valoración atribuida a cada uno de los distintos valores que toma cada parámetro de la aplicación y a partir de esta información, realizamos una selección de los valores optimizados de los atributos de forma manual.
Para la realización de este proceso se ha desarrollado un pequeño programa en **Java** que se encarga de generar tantos comandos como se le especifique en la entrada, combinando de forma aleatoria posibles valores de los parámetros del espacio de búsqueda. Estos posibles valores se guardan en *HashMaps* asociados a un número entero que actúa como índice. En este proceso se establece también el nombre del archivo de salida, el cual se nombra siguiendo una nomenclatura específica, en la que se codifican los parámetros utilizados para su generación para facilitar su decodificación posterior. El proceso de creación del nombre es el siguiente.

Figura 3.20: Nomenclatura de la imagen de salida con la información de los metadatos.

Otra función que realiza el programa es separar los comandos en lotes de 10. El motivo detrás de esta decisión es para facilitar el proceso de depuración cuando se ejecute el programa de transferencia de estilo. Estos lotes de 10 comandos son archivos `.dat`, no realizan la ejecución de los comandos que tienen escritos dentro, solo son contenedores para mantener los lotes ordenados. El encargado de la ejecución de los comandos será un *script* independiente, denominado `massive_exec.sh`. Este *script* recoge todos los archivos que contienen los comandos y los ejecuta uno a uno. Cuando comienza la ejecución de un lote, el *script* creará un archivo `.log` con su mismo nombre. En el caso de que existan comandos que al ejecutarse produzcan un error, el *script* rellenara el archivo `.log` correspondiente con dicho comando. Con esto se mantiene un control de las ejecuciones y también del progreso, para poder detener la ejecución y conocer el punto donde se termino de ejecutar para poder reanudarla posteriormente.
Para realizar todo este proceso de forma ordenada se estructura la carpeta donde se van a realizar las ejecuciones en carpetas. Debajo describimos la estructura de la carpeta del proyecto.

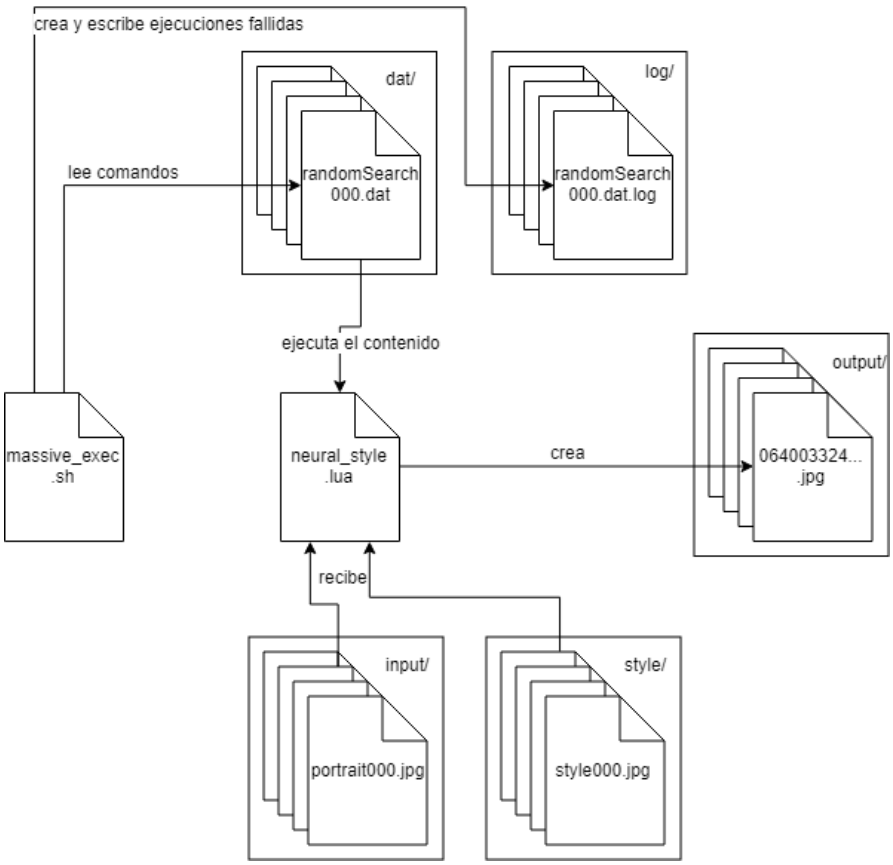


Figura 3.21: Estructura de la carpeta del proyecto con el desarrollo del proceso descrito con un diagrama.

Tras realizar aproximadamente 53 horas de ejecución se consiguieron obtener 2315 ejecuciones exitosas todas ellas imágenes de alrededor de 40kB, dándonos un tamaño total de solo 60 MB, con suficientes datos para continuar con la siguiente etapa de optimización.
Con el método de evaluación simplificado, se procesan rápidamente las imágenes moviéndolas a dos archivos distintos dependiendo de si se consideran buenas o malas a criterio del evaluador. Las imágenes que no entren en ninguno de estos dos conjuntos no se tendrán en cuenta. Tras realizar la clasificación se han obtenido 653 imágenes clasificadas como buenas y 1034 imágenes clasificadas como malas.
Una vez contamos con estas dos carpetas de imágenes clasificadas se ha creado un programa en **Java** que carga los *HashMaps* utilizados para la definición de los nombres de archivo y se utiliza como decodificador de metadatos. Este programa recibe como argumentos las dos carpetas de imágenes clasificadas, decodifica la información de los parámetros que fueron utilizados para la creación de cada imagen y con esto, por cada valor de cada parámetro calcula un *rating*.
La manera de calcular este *rating* es muy simple. Se extrae la información sobre la localización del archivo, se obtiene con ello información sobre si ha sido clasificado como buena o mala ejecución y leemos los parámetros utilizados para su creación. En el caso de que sea una imagen clasificada como buena, se suma un punto al *rating* de todos los valores utilizados que se alojan en unas estructura formada por el valor del parámetro, *rating* y frecuencia. En caso de que la imagen siendo procesada haya sido clasificada como mala, se restará uno a este *rating*.
Realizando este proceso para todas las imágenes de salida, en este caso, repitiendo el cálculo de la carpeta de imágenes buenas 5 veces y el de imágenes malas 3 veces para tratar de igualar la cantidad de datos de ambos y así normalizar los *ratings* obtenidos, podemos obtener información valiosa acerca de que valores de los parámetros brindan buenos resultados y descartar los demás.
A continuación, se presenta una de las gráficas obtenidas por este proceso. El resto de gráficas

de todos los parámetros están recogidas en el *Anexo A*.

Valor Param.	Rating	Frec.
1.E+00	-170	640
5.E+00	-49	779
1.E+01	-61	761
5.E+01	95	875
1.E+02	77	1043
5.E+02	236	1154
1.E+03	35	1115
Media:	23.29	

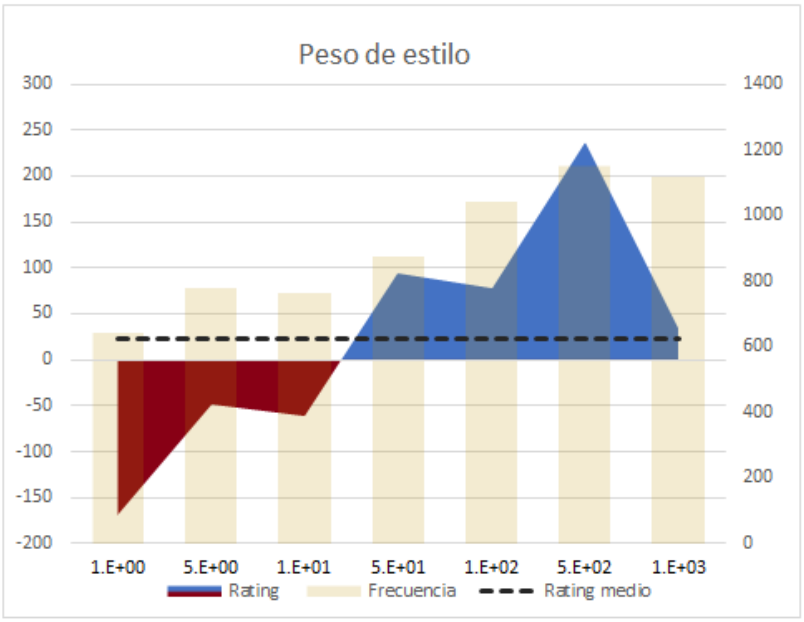


Figura 3.22: *Rating* de los distintos valores de del peso de estilo.

Por otro lado, también se realiza un *ranking* de imágenes de estilo, el cual nos muestra que imágenes de estilo han brindado mejores resultados y que imágenes de estilo generalmente resultan en una representación pobre en la imagen de salida.

Imagen	Rating	Frecuencia
11	91	259
7	62	278
24	56	254
14	41	269
26	37	223

18	-34	164
1	-36	186
12	-44	214
20	-66	156
28	-98	178

Cuadro 3.2: *Ranking* que representa el *top 5* y el *bottom 5* de imágenes de estilo.

Con estas gráficas ya generadas podemos finalizar el proceso de optimización de parámetros escogiendo manualmente aquellos valores que otorguen resultados por encima de la media. Los valores de atributos seleccionados han sido los siguientes:

Content Weight	Style Weight	TV Weight	Learning Rate	Content Layers	Style Layers	Original Colors	Pooling	Style Scale	Style Images
5.E-01	1.E+00	off	1.E-02	relu4_2,4_3	relu1_1, 2_1, 3_1, 4_1	0	avg	0.1	11
1.E+00	5.E+00	1.E-05	1.E-01	relu4_3	relu1_1, 2_1, 3_1	1	max	0.2	7
5.E+00	1.E+01	5.E-05	1.E+00	relu4_2	relu1_1, 2_1			0.4	24
1.E+01	5.E+01	1.E-04	5.E+00	relu4_1	relu1_1, 3_1, 5_1			0.6	14
5.E+01	1.E+02	5.E-04	1.E+01	relu4_1,4_2,4_3	relu2_1, 3_1, 4_1, 5_1			0.8	26
1.E+02	5.E+02	1.E-03	5.E+01	relu4_1,4_2	relu3_1, 4_1, 5_1			1	13
5.E+02	1.E+03	5.E-03	1.E+02					1.1	25
		1.E-02						1.2	5
		5.E-02							16
		1.E-01							4

Cuadro 3.3: Valores de parametros seleccionados manualmente a partir del *rating*.

Gracias a este proceso de descarte moderadamente agresivo se ha conseguido reducir el número de combinaciones por cada par de imágenes mencionado anteriormente de 4.609.920 a 768, resultando en un espacio de búsqueda notablemente más razonable, en el que 500 datos pueden ser suficientes para considerarse un subconjunto de datos representativo, haciendo posible el desarrollo de un clasificador que pueda obtener resultados adecuados.

3.6.4. Optimización auxiliar

Tras la optimización inicial descrita en el apartado anterior se realizó una batería de pruebas de verificación. Desafortunadamente, el proceso de imágenes de salida que este nuevo espacio de búsqueda proporcionaba seguía siendo relativamente bajo. Sin embargo, todas las imágenes consideradas malas habían sido clasificadas como tales por el mismo motivo, el estilo eclipsaba por completo a la imagen de contenido, problema que apuntaba a α y β (*Content Weight* y *Style Weight*) como principales promotores debido a la fórmula de *Total Loss*.
Por ello se ha realizado una segunda iteración del Random Search Optimización de manera mucho más simplificada, seleccionando el único valor de β restante tras la primera optimización con *Random Search* y variando α entre los 4 valores que superaron el primer corte por considerarse óptimos. Generando 100 nuevas imágenes en este espacio reducido de búsqueda (compuesto de solo 4 posibles variantes por cada par de imágenes) y repitiendo el proceso de clasificación manual simplificada y de generación de *ratings* (esta vez únicamente para el *Content Weight*) pudimos descartar 2 valores de α que daban consistentemente peores resultados y eran los principales culpables de la mala calidad de las imágenes de salida tras la primera iteración de optimización.

Content Weight
5.E-01
1.E+00
5.E+00
1.E+01
5.E+01
1.E+02
5.E+02

Cuadro 3.4: Valores seleccionados tras la optimización auxiliar sobre el *Content Weight*.

Con esta segunda iteración hemos bajado el número de combinaciones de los parámetros por cada par de imágenes a la mitad. Además, la calidad de los productos de los comandos ejecutados en este espacio de valores ha aumentado considerablemente. Esto fue el indicativo de que el

proceso estaba listo para pasar al último tramo del estudio.

3.6.5. Evaluación de salidas. Creación del *dataset*

Ya contando con un espacio de valores varias órdenes de magnitud más pequeño que el original y con un programa que de forma automática nos genera comandos con parámetros dentro de dicho espacio, procedimos con la generación y ejecución de 500 comandos los cuales se utilizarían posteriormente para ser evaluados y crear el *dataset* definitivo. Estas imágenes se generaron con una resolución un poco mayor a las anteriores, pasando de 400 a 512 píxeles. Tras preparar la ejecución en lotes de 10 comandos se comenzó la ejecución.

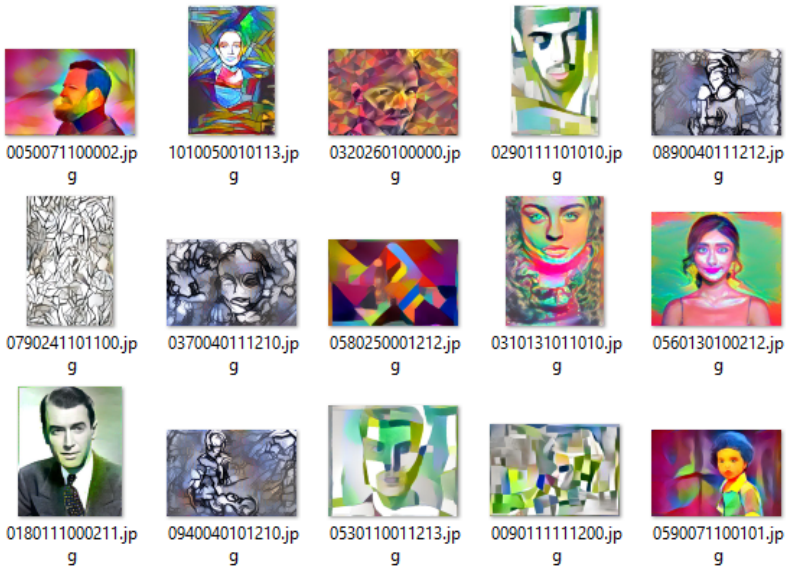


Figura 3.23: Un pequeño subconjunto de imágenes que forman el *dataset* final.

Tras aproximadamente 17 horas de ejecución el proceso de generación de imágenes termino y tuvimos acceso a las 500 imágenes que posteriormente formarían los datos del *dataset*. Por lo tanto, se pasó al proceso de evaluación manual exhaustiva con la ayuda de un programa en **Java**, el cual fue muy parecido al del etiquetado de atributos de entrada desarrollado al principio del proceso, pero adecuado a las nuevas métricas con sus respectivos rangos. También se cambió el formato de impresión en el archivo. Este programa escribiría en un archivo los valores descritos en la tabla de atributos de entrada, adjuntando como salida una puntuación de la imagen de salida que se calculó utilizando la siguiente formula:

$$
\begin{aligned}
Puntuacion &= 0{,}2D_P + 0{,}2D_A + 0{,}2C_E + 0{,}4E_S, \\
\text{donde } D_P &= \text{Discernimiento de la persona}, \\
D_A &= \text{Discernimiento de los atributos faciales}, \\
C_E &= \text{Captura de estilo}, \\
E_S &= \text{Evaluación subjetiva}
\end{aligned}
$$

Tras alrededor de 3 horas de evaluación ininterrumpida se creó el archivo de *dataset* final y se pudo proceder con el último paso del desarrollo de este estudio, La creación del modelo de regresión para la optimización de parámetros de entrada de la aplicación de transferencia de estilo.

3.6.6. Creación de los modelos

Para terminar el desarrollo y antes de pasar a la sección de experimentos se ha utilizado la herramienta de *scikit-learn* para la creación de modelos de regresión con el objetivo de obtener un modelo de regresión que sea capaz de predecir la calidad de una imagen a partir de los atributos de la imagen de contenido de entrada, imagen de estilo y los parámetros de entrada del programa. Para ello, teniendo en cuenta que el *dataset* con el que se estaba trabajando era relativamente pequeño, se utilizaron todos los modelos de regresión encontrados en la herramienta de *scikit-learn*, repitiendo algunos con distintos parámetros para intentar concebir un modelo que se ajuste a las exigencias de este estudio. Como se comentó en el apartado *1.1.3-Hipótesis*, las esperanzas de conseguir un modelo de regresión que cuente con una capacidad de predicción sobresaliente dado el tamaño tan reducido del *dataset* y la subjetividad del atributo de salida son bastante bajas. Sin embargo, gran parte del trabajo de optimización ya se realizó reduciendo el espacio de atributos mediante el *Random Search Optimization*, lo cual otorga el beneficio de la duda.
Abordando ya el proceso de creación y selección de modelos se utilizaron modelos como el *Linear Regression*, *K-Neighbors Regressor* con varios valores de k, *Support Vector Regression* con tres tipos de *kernel* distintos (rbf, polinomial y lineal), *Kernel Ridge Regressor* el cual es una variante muy parecida del *Support Vector Regression*, *Decision Tree Regressor* con profundidades 1, 2 y 5 y, por último, *Lasso Regression*.
Para elegir el modelo con el que se realizaría la predicción se sometió a todos los modelos a un proceso de validación y se calculó el Error Cuadrático Medio y el R^2 (Coeficiente de determinación) a lo largo de 10 iteraciones por cada modelo. Posteriormente se tomó el modelo con el valor medio de R^2 más cercano a 1 para realizar las predicciones de los experimentos.

Modelo	R^2	ECM
Linear Regression	0.61	1.6
Kernel Ridge	0.60	1.64
SVR lineal	0.58	1.71
SVR rbf	0.55	1.85
Decision Tree (5)	0.52	2.22

Cuadro 3.5: *Ranking* de modelos de regresión ordenados por R^2.

4

Experimentos Realizados y Resultados

Tras el largo proceso de optimización de parámetros, aquí veremos finalmente los frutos de dicho proceso, generando imágenes recomendadas por un modelo entrenado previamente.

4.1. Modelo e imágenes utilizadas

Finalmente, lo único que queda por hacer es seleccionar un conjunto de imágenes arbitrarias, generar todas las posibles combinaciones de parámetros partiendo del vector de atributos correspondiente a la imagen de contenido y, usando el modelo de regresión seleccionado devolver una predicción de las 3 mejores combinaciones de parámetros para cada imagen.
Una vez obtenidas las combinaciones con mayor nota predicha, solo queda generar los comandos asociados, ejecutarlos y validar manualmente los resultados obtenidos.
Siguiendo el *ranking* de la tabla proporcionada en la sección 3.6.6 se ha utilizado el modelo de regresión lineal de *scikit* que, teóricamente, obtendría las predicciones más precisas de acuerdo con su rendimiento en el proceso de validación de los modelos.
Para cerrar este estudio se abandona el rol de evaluador, pasando a ser el usuario. Por ello se han utilizado propias, ya que la finalidad del **Deep Style** es recreativa. Se han seleccionado las siguientes imágenes.

4.2. Proceso de preparación de ejecuciones

Como el número de combinaciones posibles de parámetros, contando las 10 imágenes de estilo posibles, es de 3840 por imagen de entrada, se ha decidido realizar la predicción de todas las combinaciones posibles. Con se han obtenido las 3 configuraciones de cada imagen que el modelo predice que obtendrán los mejores resultados en base al proceso evaluativo anterior.
Contra todo pronóstico ocurrió algo que no se había considerado. El modelo convergía a una configuración independientemente de los atributos de la imagen de contenido. Esto puede implicar dos cosas. La primera posibilidad es que, como se había comentado en el apartado de restricciones, la cantidad y capacidad de descripción de los atributos de entrada definidos es

insuficiente para este estudio. La segunda es un tanto más optimista, indicando la posibilidad que el modelo al que se ha convergido es idóneo para cualquier clase de retrato.
Por lo tanto, para descubrir cual de las dos es cierta, queda ejecutar los comandos y observar los resultados. Como se indicó al principio de este estudio, la herramienta de **Deep Style** es recreativa, por lo tanto se va a realzar el juicio esta vez dejando de lado el *rol de evaluador* y tomando en su lugar el *rol de usuario*. Con esto se pretende distanciar de la formalidad de la metodología seguida hasta ahora y pasar a contemplar las imágenes por lo que fueron destinadas a ser, obras artísticas.

4.3. Resultados

Figura 4.1: Resultado de la aplicación de estilo recomendado por el modelo de regresión lineal.

No se puede negar que los resultados son fascinantes. Antes de dar por concluido el desarrollo de este estudio se va a realizar una comparativa de parámetros utilizados de manera predeterminada por el programa frente a la configuración a la que nuestro modelo ha convergido.

Parámetro	Predeterminado	Obtenido
style_image	seated-nude.jpg	style026.jpg
content_weight	5e0	5e1
style_weight	1e2	5e2
tv_weight	1e-3	1e-2
learning_rate	1e1	1e-2
content_layers	relu4_2	relu4_3
style_layers	relu1_1,relu2_1,relu3_1, relu4_1,relu5_1	relu2_1,relu3_1,relu4_1, relu5_1
original_colors	**0**	**0**
pooling	**max**	**max**
style_scale	**1**	**1**

Cuadro 4.1: Resultado de la aplicación de estilo recomendado por el modelo de regresión lineal.

Podemos ver que es algunos parámetros como el *pooling*, escala de estilo y colores originales se han obtenido los mismos valores que los predeterminados, pero en los otros, los cuales generalmente tienen más impacto han tendido a valores completamente dispares. Este es uno de los

síntomas del *Random Search Optimization*, el cual, al no partir de una base predefinida, puede conseguir hallar resultados aceptables con combinaciones inusuales.

5

Conclusiones y trabajo futuro

Tras ver los magníficos resultados obtenidos con la configuración propuesta se observa empíricamente de forma reiterada que el estudio ha sido un éxito. Esta configuración maestra podría ser la candidata perfecta para entrenar un modelo con Normalización de Instancia como se había descrito anteriormente, puesto que parece ser un modelo bastante flexible. No obstante es un tanto decepcionante la convergencia que se ha producido hacia esta configuración, pues se deseaba ver que cambios se producían en los parámetros dependiendo de los atributos de la imagen de entrada. También resulta anticlimático por la carencia de diversidad en imágenes de estilo utilizadas, habiendo desechado 28 imágenes de las cuales 10 habían mostrado un gran potencial. Una teoría para explicar la convergencia que ocurre podría ser la falta de normalización de las entradas, que predeterminadamente esta desactivada, pudiendo resultar en la inutilidad de algún atributo. También es posible que se haya producido por la cantidad de imágenes con el estilo 26 valoradas muy positivamente, pudiendo *envenenar* la información del *dataset* obligando al modelo a tender a este estilo.
Para resolver esto, en un futuro resultaría interesante plantear:

- Explorar más a fondo el proceso realizado en este estudio, mejorando la clasificación de imágenes de entrada, añadiendo atributos a las imágenes de estilo o puliendo el proceso de evaluación de las imágenes de salida.
- También sería interesante explorar la posibilidad de automatizar el proceso de evaluación, haciendo uso de los filtros de las **CNN** para delegar la tarea de evaluación de discernimiento de sujeto y de atributos faciales a una red neuronal.
- Otro posible tema a investigar es el uso de dos estilos de manera simultánea, puesto que existen implementaciones que lo permiten, esto podría abrir muchas posibilidades pero también añadiría mucha complejidad al proceso. Esto también daría pié al proceso de segmentación de imágenes para controlar las zonas en donde un estilo es aplicado, una funcionalidad increiblemente prometedora que está disponible en la implementación de Cameron Smith[26].
- Por último, para solventar una de las limitaciones más frustrantes de este estudio, sería interesante investigar maneras de optimizar el uso de memoria en la carga y procesado

de las **CNN**, tal vez mediante alguna clase de procesamiento fraccionado, que permita usar redes que utilicen más memoria de la disponible, o incluso se alojen parcialmente en la *RAM* y la *VRAM*, dando más holgura al usuario para producir imágenes de mayor tamaño.

Glosario de acrónimos

- **CNN**: Convolutional Neural Network
- **SRCNN**: Super Resolution Convolutional Neural Network
- **SRResNet**: Super Resolution Residual Network
- **HDF5**: Hierarchical Data Format Version 5
- **ReLU**: Rectified Linear Unit
- **VGG19**: Visual Geometry Group with 19 layers
- **TV**: Total Variation
- **GPU**: Graphics Processing Unit
- **KNN**: K-Nearest Neighbors
- **ECM**: Error Cuadrático Medio
- **VRAM**: Video Random Access Memory
- **RAM**: Random Access Memory
- R^2: Coefficient of Determination

Bibliografía

[1] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[2] Katarzyna Kanska and Pawel Golinski. Using deep learning for single image super resolution. `https://blog.deepsense.ai/using-deep-learning-for-single-image-super-resolution/`, Oct 2017.

[3] Ryan Dahl, Mohammad Norouzi, and Jonathon Shlens. Pixel recursive super resolution. *CoRR*, abs/1702.00783, 2017.

[4] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4):110:1–110:11, 2016.

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[6] Otavio Good. How google translate squeezes deep learning onto a phone. *Google AI Blog*, Jul 2015.

[7] Petar Velickovic. Deep learning for complete beginners: convolutional neural networks with keras. `https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html`, Mar 2017.

[8] Rose Eveleth. How many photographs of you are out there in the world? `https://www.theatlantic.com/technology/archive/2015/11/how-many-photographs-of-you-are-out-there-in-the-world/413389/`, Nov 2015.

[9] Christian Szegedy. Building a deeper understanding of images. `https://research.googleblog.com/2014/09/building-deeper-understanding-of-images.html`, Sep 2014.

[10] Daniel Wright, Daryl Pregibon, and Diane Tang. Predicting ad quality, July 5 2007. US Patent App. 11/321,046.

[11] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. Why are deep nets reversible: A simple theory, with implications for training. *arXiv preprint arXiv:1511.05653*, 2015.

[12] Pedro Hin. r/deepdream - effects of iterations/octaves on image output. `https://www.reddit.com/r/deepdream/comments/3h3e17/effects_of_iterationsoctaves_on_image_output/`, Aug 2015.

[13] Desconocido. r/deepdream - an iterations/octave curve. `https://www.reddit.com/r/deepdream/comments/3d04jy/an_iterationsoctave_curve/`, Jul 2015.

[14] Adrienne LaFrance. When robots hallucinate. `https://www.theatlantic.com/technology/archive/2015/09/robots-hallucinate-dream/403498/`, Sep 2015.

[15] Google. Deep dream repository. `https://github.com/google/deepdream`, Aug 2015.

[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[18] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[19] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[21] Pytorch. fast neural style. `https://github.com/pytorch/examples/tree/master/fast_neural_style`, 2017.

[22] Logan Engstrom. Fast style transfer. `https://github.com/lengstrom/fast-style-transfer/`, 2016. commit c77c028.

[23] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. *CoRR*, abs/1703.07511, 2017.

[24] Rishabh Jain and Ashutosh Singh. Artistic style transfer with convolutional neural network. `https://medium.com/data-science-group-iitr/artistic-style-transfer-with-convolutional-neural-network-7ce2476039fd`, Sep 2017.

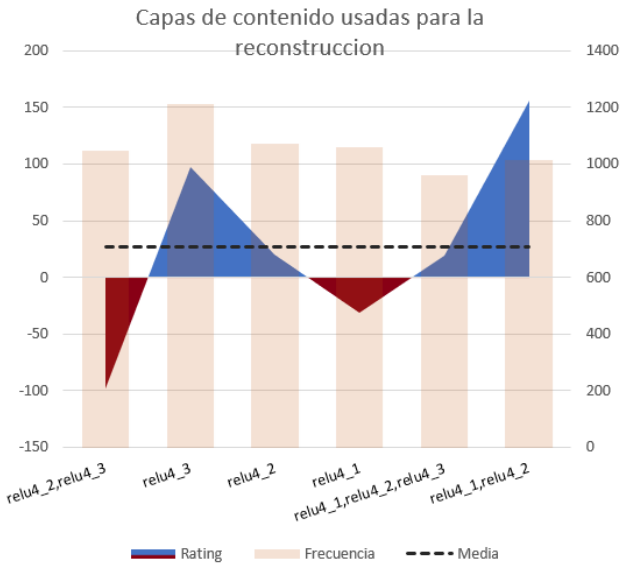
[25] Daphne Cornelisse. An intuitive guide to convolutional neural networks. `https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050`, Apr 2018.

[26] Cameron Smith. neural-style-tf. `https://github.com/cysmith/neural-style-tf`, 2016.

A

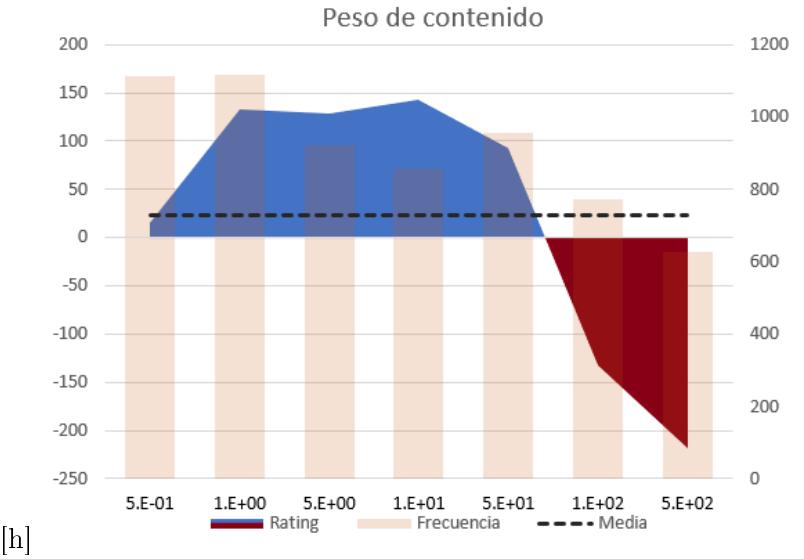
Gráficas de Parámetros seleccionados con *Random Search Optimization*

Aquí se muestran todas las gráficas obtenidas tras el proceso de *Random Search Optimization*, representando el *rating* de cada valor del parámetro, la frecuencia con la que ha aparecido en la búsqueda y la media de todos los *ratings* del parámetro:



[h]

Figura A.1: *Rating* de los distintos valores de capas para la reconstruccion de contenido.



[h]

Figura A.2: *Rating* de los distintos valores de peso de contenido.

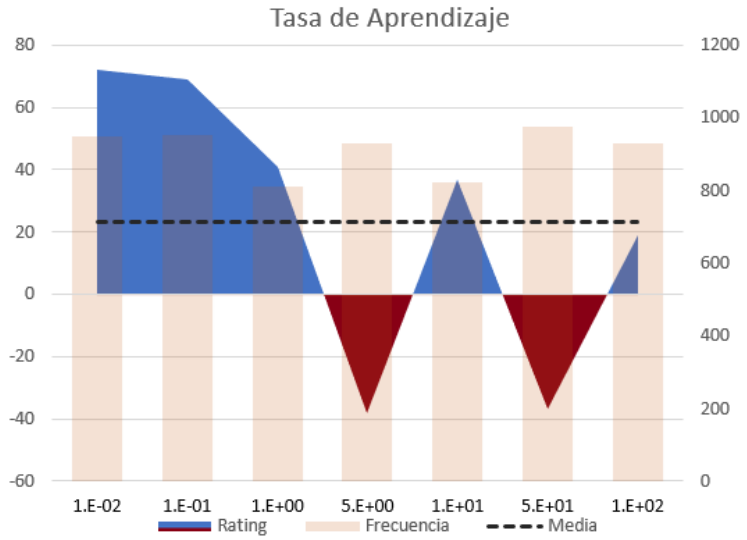


Figura A.3: *Rating* de los distintos valores de la tasa de aprendizaje.

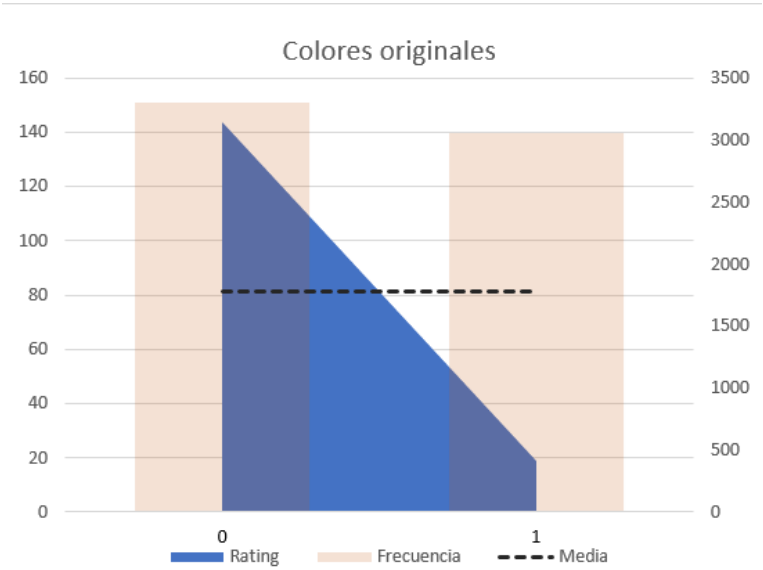


Figura A.4: *Rating* de los distintos valores del *flag* de colores originales.

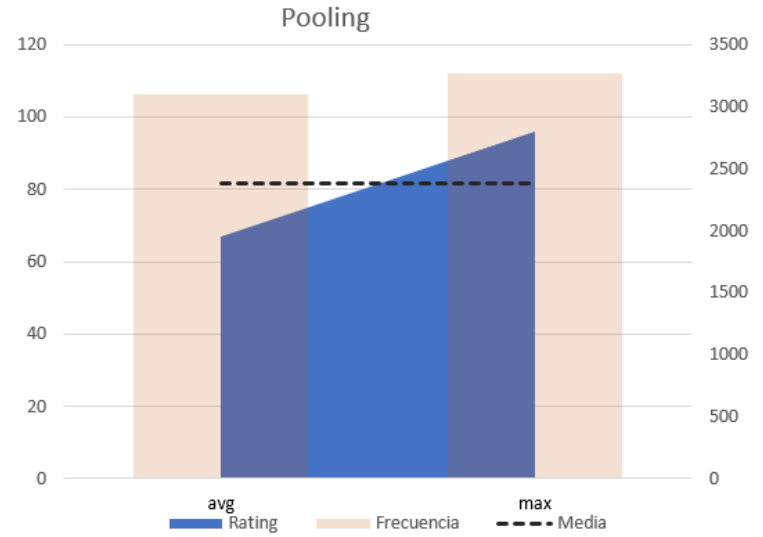


Figura A.5: *Rating* de los distintos valores de *pooling*.

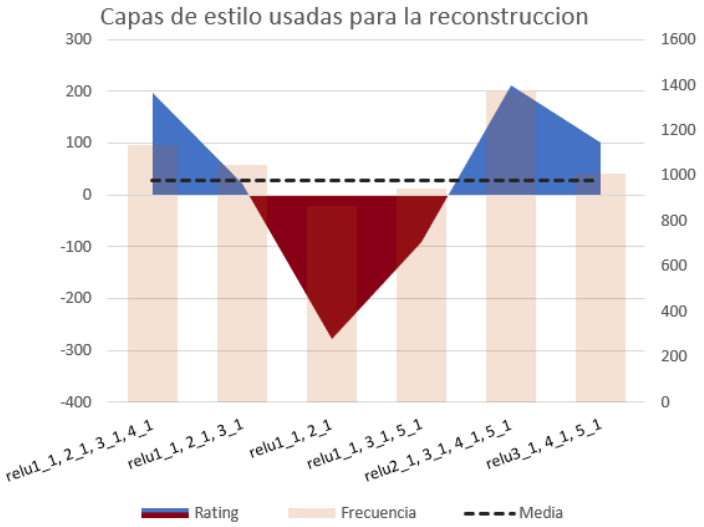


Figura A.6: *Rating* de los distintos valores de capas para la reconstruccion de estilo.

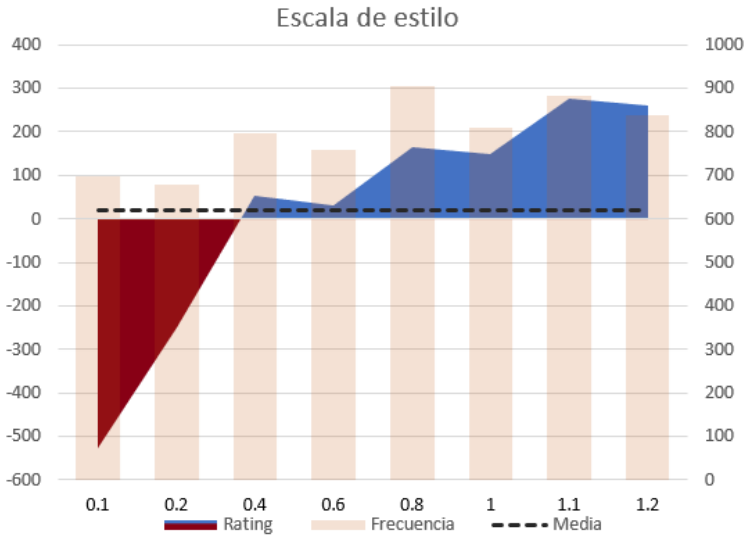


Figura A.7: *Rating* de los distintos valores de escala de estilo.

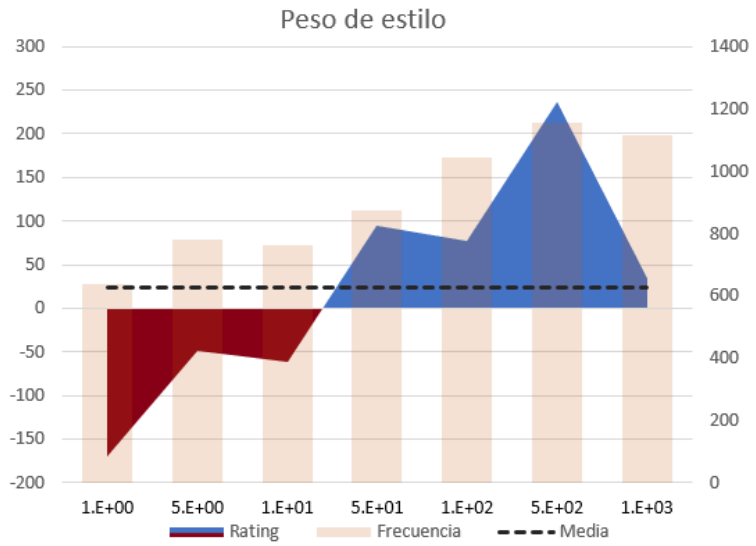


Figura A.8: *Rating* de los distintos valores de peso de estilo.

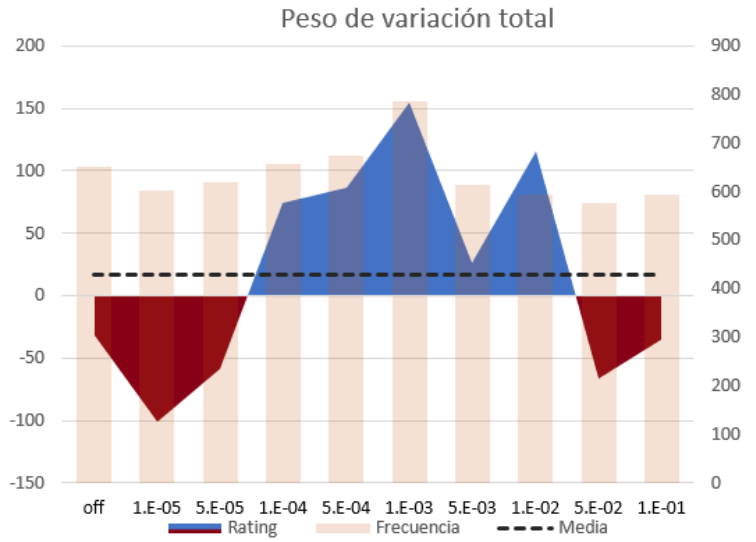


Figura A.9: *Rating* de los distintos valores de peso de variación total.

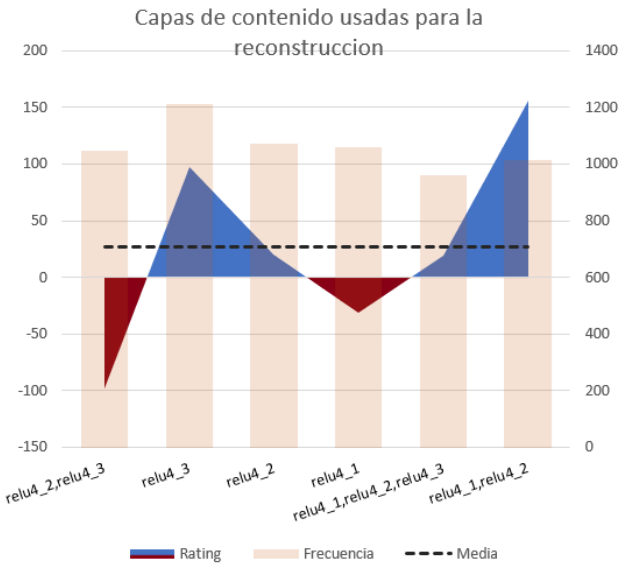


Figura A.10: *Rating* de los distintos valores de capas para la reconstruccion de contenido.